Erstellung korrekter Spezifikationen für diskrete Systeme

Roberto Ziller und Detlef Schmid
Universität Karlsruhe
Institut für Technische Informatik (ITEC)
Postfach 6980, 76128 Karlsruhe
E-Mail: {ziller,schmid}@informatik.uni-karlsruhe.de

Zusammenfassung

In dieser Arbeit werden neue Erkenntnisse zur Modellierung und zur Spezifikation von Systemen mit diskreten Zustandsräumen vorgestellt. Ein automatisches Verfahren erlaubt es, unter Berücksichtigung verschiedener Systemeigenschaften, unerwünschte Zustände während der Modellierungsphase zu erkennen und auszuschließen. Das Verfahren besteht aus einer Kombination der Überwachersynthese und der μ-Kalkül-basierten Modellprüfung. Die formale Darstellung der Systeme und deren Eigenschaften führt zu fehlerfreien Ergebnissen, vorausgesetzt, die informalen Angaben, aus denen die formale Eingabe entsteht, wurden richtig interpretiert und in die formale Eingabe übersetzt. Den Entwicklern wird ein neues, vorteilhaftes Werkzeug zur Verfügung gestellt, das sich schrittweise in bestehende Verfahren integrieren lässt, ohne bisher praktizierte Entwicklungsprozesse zu beeinträchtigen.

1 Einleitung

Der Entwurf rechnergestützter Systeme ist eine komplexe Aufgabe, die durch die Anwendung geeigneter Modellierungs- und Spezifikationstechniken erleichtert werden kann. Im Folgenden werden neue Erkenntnisse zur Modellierung und Spezifikation von Systemen mit diskreten Zustandsräumen vorgestellt, die häufig als eingebettete Systeme anzutreffen sind. Ein automatisches Verfahren erlaubt es, unter Berücksichtigung verschiedener Systemeigenschaften, unerwünschte Zustände während der Modellierungsphase zu erkennen und auszuschließen. Das Verfahren besteht aus einer Kombination bewährter Techniken aus den Gebieten der Überwachersynthese [6, 21, 22, 25, 26] und der μ-Kalkül-basierten Modellprüfung [1, 7, 9–14, 18–20, 23].

Überwachersynthese und Modellprüfung haben sich im Wesentlichen unabhängig voneinander entwickelt. Beiden gemeinsam ist die formale Darstellung der Systeme und deren Eigenschaften. Demzufolge sind die erzielten Ergebnisse immer dann fehlerfrei, wenn die informalen Angaben, aus denen die formale Eingabe entsteht, richtig interpretiert und in die formale Eingabe übersetzt wurden. Ansonsten unterscheiden sich beide Verfahren sowohl in der Herkunft, als auch in der Modellierung der Systeme und in der Zielsetzung bei der Problemstellung, wodurch sich unterschiedliche Vor- und Nachteile ergeben. In der *verallgemeinerten Überwachersynthese* [30] ergänzen sich nun beide Verfahren gegenseitig, so dass die unterschiedlichen Vorteile gemeinsam genutzt werden können und wesentliche Nachteile wegfallen. Den Entwicklern wird damit ein neues Werkzeug zur Verfügung gestellt, das viele Vorteile mit sich bringt und sich schrittweise in bestehende Verfahren integrieren lässt, ohne vorhandene Entwicklungsprozesse zu beeinträchtigen.

Im nächsten Abschnitt wird auf die Eigenschaften der angesprochenen Systeme und auf wichtige Aspekte des Systementwurfs genauer eingegangen. Abschnitt 3 fasst die Nachteile des konventionellen Entwicklungsprozesses zusammen, denen mit Hilfe der in Abschnitt 4 vorgestellten Überwachersynthese entgegengewirkt werden kann. Abschnitt 5 stellt die besonderen Vorteile dieser Methode heraus und Abschnitt 6 skizziert die damit verbundenen Veränderungen des Entwicklungsprozesses.

2 Diskrete Systeme

Die hier angesprochenen Systeme sind durch den diskreten Charakter ihres Zustandsraumes gekennzeichnet. Beispielsweise kann die Anzahl der noch verbleibenden Teile in einem Lager nur ein ganzzahliger Wert sein, wodurch sich ein diskreter Zustandsraum ergibt. Außerdem ist der Verlauf der Zeit allein nicht ausreichend, um in solchen Systemen einen Zustandswechsel herbeizuführen. Vielmehr wird hierfür ein Ereignis benötigt, so dass von *ereignisgesteuerten* Systemen die Rede ist. Im Falle des Lagers wird z.B. der Zustand durch die Entnahme oder das Hinzufügen von Teilen verändert. In dieser Arbeit sind mit dem Wort *System* ereignisgesteuerte Systeme mit diskretem Zustandsraum gemeint. Sie werden kurz *diskrete Systeme* genannt.

2.1 Systemeigenschaften

Die Eigenschaften, die in Spezifikationen diskreter Systeme anzutreffen sind, können unterschiedlicher Natur sein. Folgende Klassifikation der Eigenschaften ist auf Manna und Pnueli [16, 17] zurückzuführen:

- *Sicherheitseigenschaften* formulieren Einschränkungen, die während der gesamten Laufzeit eingehalten werden müssen. Sie werden auf Systemzustände zurückgeführt, die niemals betreten werden dürfen, wie z.B. der Zustand, in dem die Ampeln an zwei sich kreuzenden Straßen gleichzeitig grün leuchten.
- *Lebendigkeitseigenschaften* stellen die Erreichbarkeit bestimmter Zustände eines Systems sicher. Dabei wird nicht festgelegt, wie lange es dauern kann, bis ein gegebener Zustand erreicht wird. Es spielt auch keine Rolle, wie oft zu diesem Zustand zurückgekehrt werden kann. Ein Beispiel dafür ist die Forderung, dass die Initialisierung eines Systems irgendwann beendet ist.
- *Fortdauereigenschaften* beziehen sich auf Merkmale die, einmal erreicht, stabil bleiben. Ein Beispiel dazu ist ein Zwischenlager für gefertigte Teile in einer Manufaktur, das ab einer bestimmten Phase der Produktion nicht mehr leer werden darf.
- *Fairnesseigenschaften* fordern, dass Zustände mit bestimmten Eigenschaften stets erreichbar bleiben müssen, unabhängig davon, wie oft sie betreten werden. Diese Eigenschaften sind besonders für die Spezifikation reaktiver Systeme nützlich. Damit kann z.B. ausgedrückt werden, dass sich ein Benutzer bei einem System beliebig oft anmelden können soll.

Es sind nicht alle Verfahren in der Lage, mit allen Systemeigenschaften umzugehen. Zum Beispiel kann die konventionelle Überwachersynthese – gemeint ist die aus der Literatur bekannte Synthese, ohne die hier vorgestellte Verallgemeinerung – nur mit Sicherheits- und Lebendigkeitseigenschaften umgehen. Die verallgemeinerte Synthese ist hingegen in der Lage, alle Eigenschaften gleichermaßen zu behandeln [30].

2.2 Systementwurf

Die Bedingung für den korrekten Entwurf eines diskreten Systems lässt sich auf abstrakter Ebene relativ einfach formulieren. Es genügt, dass die Spezifikation für jeden erreichbaren Zustand und jedes Ereignis, das dort stattfinden kann, eine angemessene Reaktion vorsieht. Diese Aufgabe ist nicht schwierig, solange die Anzahl der zu berücksichtigen Zustände klein bleibt. Aber gerade dies ist in der Praxis normalerweise nicht der Fall. So kann beispielsweise die Tatsache, dass es einen Unterschied macht, in welcher Reihenfolge die verschiedenen Ereignisse eintreten, zu dem bekannten Problem der *Zustandsexplosion* führen.

Aus diesem Grund wurden formale Methoden, die sich zum Ziel setzten, *alle* erreichbaren Zustände eines Systems zu behandeln, oft sehr skeptisch betrachtet. Seit den Arbeiten von Burch et al. [3–5] und auch Berthet, Coudert und Madre [2] hat man in der *symbolischen Darstellung* der Zustandsräume jedoch auch für dieses Problem ein relativ effektives Mittel gefunden. Die symbolische Darstellung wurde zum Standardwerkzeug der μ-Kalkül-basierten Modellprüfung und ermöglichte die Verifikation mehrerer Protokolle und

Schaltungen, wobei auch in bereits veröffentlichten Richtlinien noch subtile Fehler gefunden wurden [8]. Die symbolische Darstellung lässt sich auch in der Überwachersynthese einsetzen [27, 28, 31–33]. Sie ist Teil der Verallgemeinerung in [30] und ermöglicht es, Zustandsräume zu behandeln, die vorher auf Grund ihrer Größe nicht im Speicher eines Rechners darstellbar waren.

Trotz dieser Fortschritte ist die Anwendung formaler Methoden, insbesondere der Überwachersynthese, noch wenig verbreitet. Dies mag mehrere Gründe haben, die nicht zuletzt in der langen Einarbeitungszeit im Vergleich zu informalen Entwicklungsprozessen liegen. Es ist aber auch ein Problem kultureller Natur. Bei dem Softwareentwurf greifen beispielsweise Entwickler in aller Regel auf ihre Erfahrung zurück oder bauen auf bereits vorhandene Lösungen auf, die auf ähnliche Weise entstanden sind. Wenn ein Programm sich dann jedoch nicht immer wie erwartet verhält, kann der Entwickler mit dem Verständnis nicht nur der Kollegen, sondern sogar der Anwender rechnen, denn die einschlägigen Probleme werden allgemein als schwer zu lösen eingestuft.

Dieser Sachverhalt ist nicht nur in der Programmierung, sondern auch in den früheren Phasen der Systementwicklung anzutreffen. Insbesondere spielt auch bei der Erstellung der Spezifikation, nach der ein System später zu bauen ist, die Erfahrung des Entwicklers eine große Rolle. Da dies für Hardware- und Softwarespezifikationen gleichermaßen gilt, können auch Hardwareschaltungen Fehler enthalten, selbst wenn die Umsetzung einer gegebenen Spezifikation in Hardware besser durch automatisierte Verfahren unterstützt wird als die Erstellung von Software. Der folgende Abschnitt listet mögliche Problemquellen auf, die dazu führen, dass komplexe Systeme in der Regel nicht fehlerfrei erstellt werden.

3 Nachteile des informalen Systementwurfs

Der allgemeine Entwicklungsprozess, der hier angesprochen werden soll, besteht aus einer Analyse- und einer Spezifikationsphase, auf welche die Implementierung und eine Testphase folgen. Von letzterer wird erwartet, dass möglichst viele Fehler, die in vorangegangenen Phasen unentdeckt blieben, noch vor der Systemfreigabe abgefangen werden. In diesem Zusammenhang sind folgende Nachteile zu erkennen:

- Der Aufwand, der mit der Behebung spät entdeckter Fehler verbunden ist, kann bekanntlich hoch sein und Projekte in Verzug geraten lassen. Dadurch werden oft noch unvollständig getestete Systeme freigegeben, um Termine einzuhalten.
- Manche Probleme sind im Voraus schwer zu erkennen, beispielsweise die Schwierigkeiten, die bei der Kommunikation zwischen verschiedenen Softwareteilen entstehen, wenn diese von unterschiedlichen Entwicklern entworfen worden sind und dabei die Spezifikation nicht immer identisch interpretiert wurde. Dann wird aus der Testphase eine letzte Implementierungsphase mit unerwartet hohem Aufwand.
- In vielen Fällen werden bereits während der Spezifikationsphase Testszenarien oder *use cases* entworfen, die einerseits zu einem besseren Verständnis des Sachverhalts führen, andererseits die Überprüfung bestimmter Funktionen sicherstellen sollen. Testszenarien werden meistens mit dem Ziel entworfen, ein erwartetes Verhalten des Systems zu bestätigen. Da das System aber gerade mit Blick auf das erwartete Verhalten aufgebaut wurde, ist erstens die Wahrscheinlichkeit, dass der Test positiv ausfällt, relativ groß. Zweitens orientiert sich die Behebung eventuell gefundener Fehler an dem gescheiterten Test. In beiden Fällen sind die Sicht des Testers und die des Entwicklers eingeengt, weil sie sich auf einen konkreten Fall und nicht auf das gesamte System konzentrieren. Fehler, die auf außergewöhnliche Ereignisse zurückzuführen sind, bleiben dadurch oft verborgen.
- Komplexe Systeme können nicht erschöpfend getestet werden, weil ihr Zustandsraum viel zu groß ist, als dass alle Zustände während der Tests durchlaufen werden könnten. Jeder nicht getestete Ablauf ist jedoch eine potentielle Fehlerquelle. In einem System, in dem es m verschiedene Ereignisse gibt, ist die Anzahl der möglichen Folgen von Ereignissen bestehend aus jeweils n Ereignissen gleich m^n. Die Wahrscheinlichkeit, einen Fehler zu finden, der erst nach mehreren Ereignissen ab dem Initialzustand

auftreten kann, nimmt infolgedessen rapide ab. Jedoch ist nicht auszuschließen, dass sich nach dem Einsatz des Systems beim Kunden gerade derartige Fehler zeigen.

- Um die Übersicht über das gesamte System nicht zu verlieren, ist es wichtig, auch die Änderungen, die zur Fehlerbehebung gemacht wurden, effizient und leicht verständlich zu dokumentieren. Dies ist insbesondere für spätere Änderungen und Erweiterungen des Systems wichtig. Dennoch stehen die Entwickler nicht selten unter so großem Zeitdruck, dass die Dokumentation vernachlässigt und damit zur Quelle neuer Fehler wird. Dies kann u.U. dazu führen, dass die Behebung eines Fehlers einen anderen hervorruft.
- In anderen Fällen wird erst nach dem Abschluss der Testphase eine Dokumentation erstellt, bei der aber dann auf wichtige Details verzichtet werden muss, weil sie auf Grund ihrer Komplexität nicht mehr genau nachvollziehbar sind. Auch damit wird die Dokumentation teilweise zur Last statt zum Hilfsmittel, und bereits gemachte Fehler werden wiederholt.

4 Die verallgemeinerte Überwachersynthese

Mit Hilfe der Überwachersynthese wird die Erstellung einer korrekten Spezifikation auf die Synthese einer Steuerung für ein diskretes System zurückgeführt. Im Folgenden werden Modellierung, Syntheseverfahren und -strategien beschrieben.

4.1 Modellierung

Das zu steuernde System wird als endlicher Automat betrachtet, in dem beliebig lange Folgen von Ereignissen ablaufen können. Je nach dem, in welcher Reihenfolge die Ereignisse auftreten, werden unterschiedliche Aufgaben vollendet. Allerdings ist es auch möglich, dass bestimmte Folgen von Ereignissen zu Verklemmungen oder anderen unerwünschten Situationen führen, und diese gilt es mit Hilfe einer Steuerung zu vermeiden. Wichtig ist, dass der Automat, mit dem das System modelliert wird, sämtliche physikalisch möglichen Ereignisfolgen darstellt, also auch die unerwünschten. Die übliche Bezeichnung für diesen Automaten ist $\mathcal{A}_{\mathcal{P}}$.

Außer der Systembeschreibung $\mathcal{A}_{\mathcal{P}}$ wird eine Spezifikation für das gewünschte Verhalten benötigt. Auch dieses wird mit Hilfe eines Automaten modelliert. Dazu wird ein Hilfsautomat $\mathcal{A}_{\mathcal{E}}$ erstellt, und mit diesem das Produkt mit dem Automaten $\mathcal{A}_{\mathcal{P}}$ gebildet. Das Produkt $\mathcal{A}_{\mathcal{P}} \times \mathcal{A}_{\mathcal{E}}$ ist die Spezifikation. Auf Grund der Eigenschaften des Produkts ist die Menge der möglichen Ereignisfolgen in der Spezifikation eine Untermenge des physikalisch möglichen Verhaltens.

Der Grundgedanke des Ramadge-Wonham-Modells [21, 22, 25] ist es, das System mit Hilfe eines *Überwachers* wie in Abbildung 1 zu steuern. Der Überwacher ist ebenfalls ein endlicher Automat, der jedes Ereignis im System verfolgt und darauf mit einer Steuerungsaktion reagiert. Die Steuerungsaktion besteht darin, dem System nach jedem Zustandswechsel mitzuteilen, welche Ereignisse in dem neu betretenen Zustand des Überwachers vorgesehen sind. Per Konstruktion ist dies stets eine Untermenge der Ereignisse, die von der Spezifikation im aktuellen Zustand des Systems zugelassen sind. Unter der Voraussetzung, dass das System das nächste Ereignis aus dieser Untermenge wählt, wird die Spezifikation für das gewünschte Systemverhalten eingehalten.

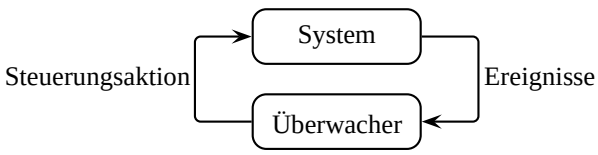

Abbildung 1: Die Überwachung eines diskreten Systems

4.2 Die Berechnung des Überwachers

Im Idealfall dient die Spezifikation $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ selbst als Überwacher. Dies ist jedoch nicht immer möglich, weil es Ereignisse gibt, die sich nicht unterbinden lassen. Beispiele hierfür sind der Ablauf eines Zeitglieds, die Ankunft einer Meldung in einem Kommunikationssystem oder das Ansprechen eines Sensors. Hingegen lassen sich beispielsweise Befehle zum Ein- und Ausschalten von Geräten oder zum Senden von Meldungen falls notwendig verhindern. Dementsprechend werden die Ereignisse in *steuerbar* bzw. *nicht steuerbar* unterteilt. Der Einsatz einer Spezifikation als Überwacher ergibt folglich nur dann einen Sinn, wenn diese in keinem Zustand versucht, ein nicht steuerbares Ereignis zu verhindern.

Es ist aber auch zu berücksichtigen, dass sowohl der Automat $\mathcal{A}_\mathcal{P}$ als auch $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ sehr große Zustandsräume haben können. Sie werden in der Regel nicht in einem Guss erstellt, sondern mit Hilfe der parallelen Komposition und des Produkts kleinerer, überschaubarer Automaten berechnet. Dabei geht der Überblick über die einzelnen Zustände rasch verloren, und so ist es keine Seltenheit, dass Anforderungen, die keineswegs abwegig erscheinen, doch in einigen Zuständen versuchen, nicht steuerbare Ereignisse zu verhindern. Diese Zustände der Spezifikation werden *verbotene Zustände* genannt.

Es ist ebenso möglich, dass die Spezifikation Pfade enthält, die zu keiner vollendeten Aufgabe führen. Dies ist entweder dann der Fall, wenn noch keine Aufgabe vollendet wurde und keine weiteren Ereignisse mehr möglich sind *(deadlock)* oder dann, wenn das System sich nur noch innerhalb einer Zustandsmenge bewegen kann, die zu keiner vollendeten Aufgabe führt (*livelock*).

Darüber hinaus kann es noch Zustände geben, von denen aus nur noch ein Teil der Aufgaben des Systems vollendet werden kann, oder auch solche, von denen aus nur eine begrenzte Anzahl von Wiederholungen eines Vorgangs möglich ist. Es gibt aber Vorgänge, die unendlich oft möglich sein sollten, wie z.B. das Abschicken von Meldungen in einem Kommunikationssystem oder die Anmeldung von Benutzern an einem Rechner.

Aus dieser Beschreibung ist ersichtlich, dass die Spezifikation $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ allein nicht ausreicht, um zu einem vollständigen Überwacher zu gelangen. Es ist jedoch möglich, aus $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ einen Überwacher zu berechnen. Dieser Vorgang wird als *Überwachersynthese* bezeichnet. Die konventionelle Überwachersynthese besteht darin, verbotene oder zu Verklemmungen führende Zustände aus $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ zu entfernen. Der Gedanke dabei ist, den resultierenden Automaten als Überwacher einzusetzen, sofern die Einschränkungen, die der Spezifikation durch den Syntheseprozess auferlegt wurden, akzeptabel sind.

Damit ist die konventionelle Synthese in der Lage, Sicherheits- und Lebendigkeitseigenschaften zu berücksichtigen, nicht aber Fortdauer- oder Fairnesseigenschaften. Der Unterschied zwischen der konventionellen und der verallgemeinerten Überwachersynthese besteht darin, dass in der letzteren die Anforderungen an die Berechnung des Überwachers nicht mehr fest, sondern vom Entwickler wählbar sind. Verbotene Zustände werden nach wie vor entfernt, aber an die Stelle der Verklemmungsfreiheit können strengere Anforderungen treten. So ist z.B. ein System, das nach einer bestimmten Betriebszeit nur noch einen Teil seiner Aufgaben erfüllen kann, im Sinne der konventionellen Synthese immer noch verklemmungsfrei. In dem verallgemeinerten Ansatz ist es aber möglich, genau anzugeben, welche Aufgaben stets noch zu vollenden sein müssen, damit das System als verklemmungsfrei gilt.

Für die Verallgemeinerung der Überwachersynthese wurde in [30] eine Transformation definiert, mit der aus dem Automaten $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ eine spezielle Kripke-Struktur $\mathcal{K}_{\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}}$ erstellt wird. Zusätzlich wurde einer der konventionellen Synthesealgorithmen in ein μ-Kalkül-Gleichungssystem übersetzt. Die Lösung eines solchen Gleichungssystems besteht aus einer Untermenge des Zustandsraums einer gegebenen Kripke-Struktur. In diesem Fall sind die Kripke-Struktur $\mathcal{K}_{\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}}$ und das Gleichungssystem so aufeinander abgestimmt, dass dessen Lösung, wenn auf den Automaten $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ abgebildet, den gleichen Überwacher hervorbringt wie der ursprüngliche Algorithmus. Dieser Zusammenhang wird in Abbildung 2 dargestellt.

Das Gleichungssystem hat außerdem die vorteilhafte Eigenschaft, dass die Forderungen nach Steuerbarkeit und Verklemmungsfreiheit in getrennte Gleichungen zerfallen. Damit ist es möglich, an die Stelle der Gleichung für die Verklemmungsfreiheit beliebige andere μ-Kalkül-Gleichungen zu setzen. Der Anwender kann die notwendigen Gleichungen mit Hilfe temporallogischer Ausdrücke erzeugen [23, 30] und damit die Kluft

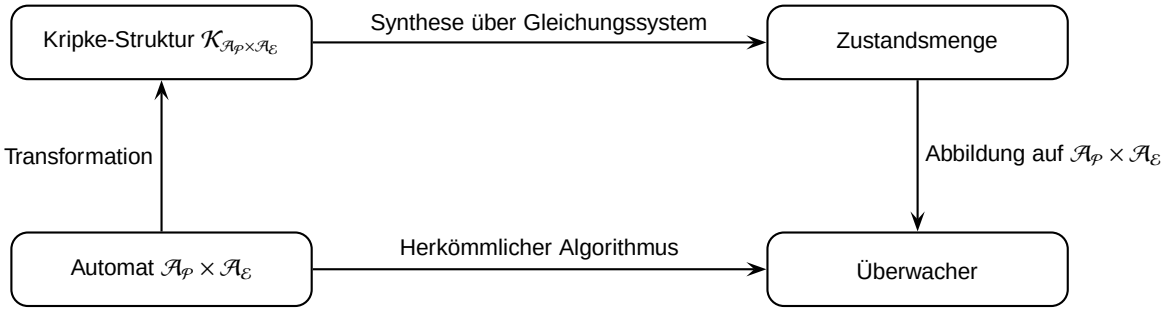

Abbildung 2: Die Übersetzung der Überwachersynthese in den μ-Kalkül

zwischen der menschlichen Intuition und der teilweise komplizierten Syntax der μ-Kalkül-Ausdrücke überbrücken. Da der μ-Kalkül eine Obermenge gängiger Temporallogiken wie CTL, LTL und CTL* ist, werden alle Systemeigenschaften, die in diesen Logiken ausdrückbar sind, gleichermaßen behandelt. Dies trifft insbesondere für Fortdauer- und Fairnesseigenschaften zu, die in der konventionellen Synthese nicht handhabbar sind.

4.3 Spezifikationsstrategien

Wie in Abschnitt 4.2 erwähnt, können auch dann verbotene Zustände in Spezifikationen entstehen, wenn die Anforderungen, die an das System gestellt werden, auf den ersten Blick harmlos erscheinen. Als Beispiel sei die Steuerung einer Telefonnummernauskunftzentrale *(call center)* [24] genannt, die in [29, 30] im Detail analysiert wird.

In vielen Fällen werden unerwünschte Zustände auch absichtlich erzeugt. Oft ist es leicht, die Zustände anzugeben, die ein System während des Betriebs *nicht* erreichen sollte, ohne jedoch absehen zu können, welche Bedingungen dafür eingehalten werden müssen. Dies ist z.B. bei der Berechnung einer Gewinnstrategie der Fall. Aus theoretischer Sicht kommt die Aufgabe, ein reaktives System so zu steuern, dass ein bestimmter Zustand erreicht wird, der Anwendung einer Gewinnstrategie für ein Zweipersonenspiel gleich. Der Automat $\mathcal{A}_\mathcal{P}$ für die Systembeschreibung gibt dann alle möglichen Spielabläufe wieder. Darunter befinden sich welche, die zu dem gewünschten Zustand führen (die Steuerung „gewinnt“) und auch solche, die zu unerwünschten Zuständen führen (die Steuerung „verliert“). Auf abstrakter Ebene lautet die Spezifikation „die Steuerung soll nicht verlieren“. In der Spezifikation $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ werden daraufhin die Zustände, in denen die Steuerung „verlieren“ würde, absichtlich als Verklemmungen dargestellt. Die Syntheseprozedur entfernt dann diese Verklemmungen und alle nicht steuerbaren Zustände, die dazu führen könnten. Übrig bleibt die Gewinnstrategie für das „Spiel“, nach der das System zu steuern ist. Ein Beispiel für diese Art der Anwendung ist die Synthese einer Gewinnstrategie für das Nim-Spiel in [28, 30].

Es genügt also, die Spezifikation so zu erstellen, dass unerwünschte Zustände entweder in dieser gar nicht auftreten, oder aber so gekennzeichnet sind, dass sie von der Syntheseprozedur erkannt und ausgeschlossen werden. In der Regel können Zustände, die eindeutig vermieden werden sollen, bereits bei der Erstellung der Spezifikation ausgeschlossen werden. Hingegen werden Zustände, die an sich nicht unerwünscht sind, aber Pfade zu solchen Zuständen besitzen, erst durch die Syntheseprozedur erkannt. Nach welchem Kriterium dies genau geschieht, hängt von dem Gleichungssystem ab, das der Anwender für seine Zwecke erstellt.

5 Vorteile der Überwachersynthese

Der bedeutendste Vorteil der Überwachersynthese besteht in der Übersetzung abstrakter Spezifikationen, die ein gewünschtes Verhalten des Systems beschreiben, in konkrete Baupläne. Dies kann mit den folgenden Beispielen verdeutlicht werden.

Beispiel: Zu erstellen sei ein Kommunikationsprotokoll, mit dem Meldungen von einem Sender zu einem Empfänger über ein verlustbehaftetes Medium zu übertragen sind. Geht eine Meldung verloren, muss der Sender sie wiederholen und gegebenenfalls mehrere Kopien einer Meldung abschicken. Die informale Spezifikation des Protokolls wird deshalb vorschreiben, dass letztendlich keine Meldungen verloren gehen und auch keine mehrfachen Kopien am Empfänger ankommen dürfen. Dies ist zwar eine präzise Definition von dem, was gewünscht wird, aber wenig hilfreich, wenn es darum geht, diese Regeln in eine Schaltung oder in ein Programm umzusetzen. Der Einsatz der Überwachersynthese kann an dieser Stelle bedeutende Hilfe leisten. Sender, Medium und Empfänger werden zunächst mit endlichen Automaten modelliert, so dass eine Systembeschreibung entsteht, die alle möglichen Vorgänge enthält – also auch den Verlust und die mehrfache Auslieferung einer Meldung. Die Spezifikation besagt dann, dass der Empfänger genau eine Kopie von jeder Meldung empfangen muss. Daraus berechnet die Syntheseprozedur einen Überwacher, aus dessen Transitionsrelation die zulässigen Ereignisse für jeden erreichbaren Zustand hervorgehen. Diese Information hat nun die Form, die der Entwickler braucht, um Hardware oder Software zu erstellen.

Weiteres Beispiel: Der Gewinn an Information, den die Überwachersynthese mit sich bringt, kann auch anhand der in Abschnitt 4.3 beschriebenen Synthese einer Gewinnstrategie verdeutlicht werden. Die Anforderung „die Steuerung soll nicht verlieren“ ist zu abstrakt, als dass sie direkt in eine Schaltung bzw. in ein Programm umgesetzt werden könnte. Hingegen enthält der Überwacher genaue Anweisungen, wie die Lösung zu implementieren ist.

Das Ergebnis der Synthese ist ein endlicher Automat, der Auskunft darüber gibt, *was* eigentlich zu implementieren ist. Es gibt verschiedene Möglichkeiten, diese Information zu nutzen. Diese reichen von dem einfachen Auslesen der Transitionsrelation, um die manuelle Programmierung zu unterstützen, bis zur voll automatisierten Übersetzung des Überwachers in Hardware oder Software [15]. Dadurch wird eine schrittweise und risikoarme Einführung der Überwachersynthese in bestehende Entwicklungsprozesse ermöglicht.

Darüber hinaus ergeben sich noch andere Vorteile:

- Nähe zur informalen Sprache: Am Beginn eines jeden Entwicklungsprozesses steht normalerweise eine Beschreibung des Systems in natürlicher Sprache (z.B. Deutsch oder Englisch). Die darin enthaltenen Anforderungen müssen ins Formale übersetzt werden, und von diesem Schritt hängt die Korrektheit des Ergebnisses ab. Durch die Möglichkeit, Anforderungen zunächst in temporallogische Ausdrücke und dann in den μ-Kalkül zu übersetzen, kann die verallgemeinerte Überwachersynthese nahtlos an die informale Spezifikation anknüpfen.

- Größere Flexibilität: Änderungen an den Spezifikationen, z.B. um auf neue Wünsche des Auftraggebers einzugehen, können mit geringem Zeitaufwand umgesetzt werden. Ist der gesamte Prozess automatisiert, kann eine neue Version des gesamten Systems per Knopfdruck erstellt werden.

- Verringerte Testzeit: Es besteht keine Notwendigkeit, das aus dem Überwacher abgeleitete System auf Eigenschaften zu testen, die in die Synthese eingeflossen sind. Damit verlagert sich der Testschwerpunkt auf die Übereinstimmung der informalen Anforderungen mit der realisierten Funktionalität. Werden Unstimmigkeiten festgestellt, werden die Automaten bzw. das Gleichungssystem angepasst und ein neues System generiert. Das Testen bleibt auf eine höhere Ebene beschränkt.

- Auswirkungen von späteren Änderungen: In der traditionellen Systementwicklung sind spät entdeckte Fehler in der Regel mit hohen Kosten verbunden. Mit der automatisierten Synthese wird jede Änderung auf der Ebene der Automaten und des Gleichungssystems vorgenommen und danach das gesamte System neu erstellt. Alle Abhängigkeiten werden automatisch berücksichtigt, so dass sich keine Folgefehler einschleichen können.

- Steigerung der Qualität: Der Entwicklungsprozess wird nicht nur agiler, sondern auch zuverlässiger. Durch die Einbeziehung des gesamten Zustandsraums werden Fehler bei der Implementierung ausgeschlossen. Als mögliche Fehlerquellen bleiben nur eine falsche Interpretation der anfänglichen Systembeschreibung und Hardwarefehler zur Laufzeit übrig.

6 Auswirkungen auf den Entwicklungsprozess

Eine Umstellung des Entwicklungsprozesses auf die voll automatisierte Überwachersynthese würde einen tiefen Einschnitt in die traditionellen Entwicklungsmethoden mit sich bringen. Bekanntlich ist es bereits schwierig, jemanden, der sich in einer bestimmten Programmiersprache oder mit einem bestimmten Programmsystem gut auskennt, von einem Wechsel zu überzeugen. Der Umstieg auf eine völlig andere Entwicklungsmethode ist deshalb nur schrittweise durchführbar. Um die Einführung der Überwachersynthese zu erleichtern, kann sie zunächst nur als Hilfswerkzeug herangezogen werden. Der Entwickler entscheidet dann anhand der Transitionsrelation des Überwachers, *was* zu implementieren ist. Die Implementierung selbst kann davon zunächst unberührt bleiben und erst später automatisiert werden. Durch die gesammelten Erfahrungen sollte eine Eigendynamik entstehen, die den Einsatz der neuen Methoden steuert und das Vertrauen in sie stärkt. Außer der in Abschnitt 5 beschriebenen Verlagerung der Testschwerpunkte würden dann Merkmale wie die folgenden den veränderten Entwicklungsprozess kennzeichnen:

- Systementwurf auf höherer Ebene: Ähnlich wie bei dem Übergang von Assembler auf höhere Programmiersprachen findet ein Übergang von diesen auf eine höhere Ebene statt. Programme oder auch Hardwarebeschreibungen werden zum Zwischenformat, das jedoch nicht mehr von Hand erstellt wird. Sie bleiben trotzdem noch lesbar, was anfangs ein wichtiger Faktor zur Vertrauensbildung sein kann. Es ist jedoch vorstellbar, dass dieses Bedürfnis nach und nach verschwindet, genauso wie es heute kaum noch einen Grund dazu gibt, den Assemblercode zu untersuchen, der von einem Übersetzer *(compiler)* erzeugt wurde.

- Die Rolle der Dokumentation: Kommentare in Programmen können entweder automatisch mit eingefügt werden oder ganz wegfallen, besonders wenn diese zu dem o.g. Zwischenformat werden. Viel wichtiger ist es, festzuhalten, *wie* die verwendeten Automaten und Gleichungssysteme entstehen. Da hier eine organisierte Vorgehensweise klare Vorteile mit sich bringt, wird die Dokumentation zu einem nützlichen Werkzeug, dass insbesondere während des Entwicklungsprozesses eingesetzt und gepflegt wird – im Gegensatz zur Programmdokumentation, die nicht selten erst nach der Fertigstellung des Systems geschrieben wird.

- Werkzeuge und Schulungsmaßnahmen: Es entsteht ein Bedarf an neuen Programmen zur Unterstützung der Synthese. Denkbar sind auch Bibliotheken mit Lösungen gängiger Teilprobleme sowie die Integration mit weiteren Spezifikationsverfahren wie z.B. *statecharts* oder UML. Entsprechend würden sich auch die Schulungen zur Entwicklertätigkeit verändern.

7 Zusammenfassung

In dem konventionellen Entwicklungsprozess für Hard- und Software ist die Umsetzung einer Spezifikation in ein testbares Produkt mit hohem Aufwand verbunden. In Abhängigkeit der Fehler, die in der Testphase entdeckt werden, muss diese Umsetzung mehrmals wiederholt werden. Dadurch entstehen verschiedene Nachteile, die erhebliche Verzögerungen und Qualitätsverluste zur Folge haben können. Mit der verallgemeinerten Überwachersynthese lässt sich ein neuer Entwicklungsprozess realisieren, in dem die Arbeit sich auf höhere Abstraktionsebenen konzentriert und Änderungen sauber und effizient durchzuführen sind. Das neue Verfahren knüpft an die informalen Anforderungen an, die am Beginn des Entwicklungsprozesses stehen. Sind diese formal erfasst, geschieht der restliche Ablauf automatisch. Programme und Hardwarebeschreibungen auf höherer Ebene werden zum Zwischenformat und sowohl Testaufwand als auch Änderungen am System konzentrieren sich auf die Übereinstimmung zwischen informaler und formaler Darstellung der Anforderungen.

Der neue Entwicklungsablauf erfordert ein tiefgreifendes Umdenken, das nicht schlagartig zu erreichen ist. Eine schrittweise Einführung der Überwachersynthese ist jedoch möglich, indem die formal erzeugten Spezifikationen zunächst nur als Hilfsmittel herangezogen und wie gehabt umgesetzt werden. Dies ist auch deshalb

angebracht, weil sich nicht jedes Problem mit formalen Methoden lösen lässt. Es sollte sich für jeden Anwender Schritt für Schritt herausstellen, wie die Überwachersynthese im Einklang mit anderen Methoden einzusetzen ist. Ziel dieser Arbeit ist es, solche Überlegungen zu unterstützen und zu erleichtern.

Literatur

[1] A. Arnold and P. Crubille. A linear algorithm to solve fixed-point equations on transition systems. *Information Processing Letters*, 29(2):57–66, 1988.

[2] C. Berthet, O. Coudert, and J.C. Madre. New ideas on symbolic manipulations of finite state machines. In *International Conference on Computer Design (ICCD)*, pages 224–227. IEEE Computer Society, 1990.

[3] J.R. Burch, E.M. Clarke, K.L. McMillan, and D.L. Dill. Sequential circuit verification using symbolic model checking. In *Design Automation Conference (DAC)*, pages 46–51, Los Alamitos, CA, June 1990. ACM.

[4] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Symposium on Logic in Computer Science (LICS)*, pages 1–33, Washington, D.C., June 1990. IEEE Computer Society.

[5] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, June 1992.

[6] C.G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, Boston, U.S.A., 1999. ISBN 0-7923-8609-4.

[7] E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In D. Kozen, editor, *Workshop on Logics of Programs*, volume 131 of *LNCS*, pages 52–71, Yorktown Heights, New York, May 1981. Springer.

[8] E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking*. MIT, London, England, 1999.

[9] R. Cleaveland, M. Klein, and B. Steffen. Faster model checking for the modal μ-calculus. In G.v. Bochmann and D. Probst, editors, *Conference on Computer Aided Verification (CAV)*, volume 663 of *LNCS*, pages 410–422, Montreal, June/July 1992. Springer.

[10] R. Cleaveland and B. Steffen. A linear-time model checking algorithm for the alternation-free μ-calculus. In K.G. Larsen and A. Skou, editors, *Conference on Computer Aided Verification (CAV)*, volume 575 of *LNCS*, pages 48–58, Aalborg, Denmark, July 1991. Springer.

[11] A. Dicky. An algebraic and algorithmic method of analyzing transition systems. *Theoretical Computer Science*, 46:285–303, 1986.

[12] E.A. Emerson and C.-L. Lei. Efficient model checking in fragments of the propositional μ-calculus. In *Symposium on Logic in Computer Science (LICS)*, pages 267–278, Washington, D.C., 1986. IEEE Computer Society.

[13] D. Kozen. Results on the propositional μ-calculus. In *Colloquium on Automata, Languages and Programming (ICALP)*, pages 348–359, 1982.

[14] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, December 1983.

[15] P. Malik. *From Supervisory Control to Nonblocking Controllers for Discrete Event Systems*. PhD thesis, Universität Kaiserslautern, 2003.

[16] Z. Manna and A. Pnueli. The anchored version of the temporal framework. In *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, volume 354 of *LNCS*, pages 428–437, Noordwigherhout, Netherland, May/June 1988. Springer.

[17] Z. Manna and A. Pnueli. A hierarchy of temporal properties. In *Symposium on Principles of Distributed Computing*, pages 377–408, 1990.

[18] V. Pratt. A decidable μ-calculus. In *Symposium on Foundations of Computer Science (FOCS)*, pages 421–427, New York, 1981. IEEE Computer Society.

[19] J.P. Quielle and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *Symposium in Programming*, 1981.

[20] J.P. Quielle and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *Symposium in Programming*, volume 137 of *LNCS*, pages 337–351, New York, 1982. Springer.

[21] P. J. Ramadge and W. M. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81–98, 1989.

[22] P.J. Ramadge and W.M. Wonham. Supervisory control of a class of discrete event processes. *SIAM Journal of Control and Optimization*, 25(1):206–230, 1987.

[23] K. Schneider. *Verification of Reactive Systems – Formal Methods and Algorithms*. Texts in Theoretical Computer Science (EATCS Series). Springer, 2003. ISBN 3-540-00296-0.

[24] M. Seidl, T. Kleinert, J. Wagner, and B. Plannerer. Systematischer Steuerungsentwurf zur Kontrolle lastintensiver Call-Center. *at - Automatisierungstechnik*, 50:19–27, June 2002.

[25] W. M. Wonham. Supervisory control of discrete-event systems. Technical report, Dept. of Electrical and Computer Engineering, University of Toronto, Jul. 2005. ECE 1636F/1637S 2005-06, http://www.control.utoronto.ca/DES.

[26] W.M. Wonham and P.J. Ramadge. On the supremal controllable sublanguage of a given language. *SIAM Journal of Control and Optimization*, 25(3):637–659, May 1987.

[27] Z.H. Zhang and W.M. Wonham. STCT: An efficient algorithm for supervisory control design. In *Symposium on Supervisory Control of Discrete Event Systems (SCODES)*, Paris, France, July 2001.

[28] R. Ziller. Finding bad states during symbolic supervisor synthesis. In *GI/ITG/GMM Workshop: Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen*, pages 209–218, Tübingen, Germany, 25-27 February 2002.

[29] R. Ziller. An Application of Generalized Supervisor Synthesis to the Control of a Call Center. In *Forum on Specification and Design Languages*, pages 429–440, Lausanne, Switzerland, September 2005.

[30] R. Ziller. *Eine Verallgemeinerung der Überwachersynthese mit Hilfe des μ-Kalküls [online]*. Dissertation, Universität Karlsruhe, Fakultät für Informatik, Deutschland, 2005. http://www.ubka.uni-karlsruhe.de/cgi-bin/psview?document=2005/informatik/21.

[31] R. Ziller and K. Schneider. A generalized approach to supervisor synthesis. In *Formal Methods and Models for Codesign (MEMOCODE)*, pages 217–226, Mont Saint-Michel, France, 2003. IEEE Computer Society.

[32] R. Ziller and K. Schneider. A μ-calculus approach to supervisor synthesis. In R. Drechsler, editor, *GI/ITG/GMM Workshop: Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen*, pages 132–143, Bremen, Germany, 24-26 February 2003. Shaker.

[33] R. Ziller and K. Schneider. Combining Supervisor Synthesis and Model Checking. *ACM Transactions on Embedded Computing Systems*, 4(2):331–362, May 2005.