Trace-basiertes Testen von nebenläufigen reaktiven Systemen*

Roland Schatz
roland@rschatz.org

Abstract: Speicherprogrammierbare Steuerungsprogramme sind Programme, die zur Steuerung von Maschinen eingesetzt werden. SPS-Programme bestehen aus einem Regelungs- und einem reaktiven Steuerungsteil. Während die Analyse von Regelungen ein etabliertes Forschungsgebiet ist, wurde der Analyse des reaktiven Verhaltens von SPS-Programmen bisher wenig Beachtung geschenkt. Gleichzeitig ist aber die Analyse von reaktiven Systemen eine komplexe und anspruchsvolle Aufgabe.

Diese Arbeit präsentiert einen Formalismus zur Spezifikation und Analyse des reaktiven Verhaltens von SPS-Programmen. Aufbauend auf ein Verfahren zur Aufzeichnung und deterministischen Wiedergabe von SPS-Programmen wird eine auf Hybriden Automaten basierende Spezifikationssprache vorgestellt, mit der sowohl Testen als auch die Analyse von Aufzeichnungen ermöglicht wird.

1 Einleitung

Speicherprogrammierbare Steuerungsprogramme (kurz: SPS-Programme) sind Programme zur Steuerung von Maschinen wie zum Beispiel Fertigungsanlagen oder Roboter. Sie interagieren mit der physikalischen Welt, indem sie laufend Sensoreingaben lesen und Aktuatorausgaben schreiben. SPS-Programme sind lang laufende Anwendungen die komplexes zeitliches Verhalten aufweisen.

SPS-Programme bestehen aus zwei Ebenen: Einerseits implementieren sie Regelkreise, die die grundlegenden Aktivitäten des angeschlossenen Geräts regeln. Darüber hinaus gibt es noch eine Entscheidungsebene, die Zustandsübergänge steuert, die einzelnen Regler-Bausteine ein- und ausschaltet und das Gesamtverhalten des Systems koordiniert. Solche Systeme haben komplexes reaktives Verhalten.

Reaktive Systeme sind diskrete Systeme, die laufend mit ihrer Umgebung interagieren und auf externe und interne Stimuli reagieren [HP85]. Im speziellen Fall von SPS-Programmen beschreibt das reaktive Verhalten, wie das Steuerungsprogramm auf bestimmte Eingabewerte von den Sensoren reagiert (z.B. das Abschalten eines Motors an der Endposition). Im Vergleich dazu besteht der Regler nur aus einer mathematischen Formel die die gewünschte Geschwindigkeit des Motors regelt.

Während die Analyse von Reglern ein gut erforschtes Gebiet mit etablierten Anwendungen in der Praxis ist, wurde der Analyse des reaktiven Verhaltens von SPS-Programmen

*Englischer Titel der Dissertation: "Trace-based Testing of Multi-threaded Reactive Systems"

bisher wenig Aufmerksamkeit in Praxis oder Forschung geschenkt. Auf der anderen Seite ist es weitgehend anerkannt, dass Entwurf, Entwicklung und Analyse von reaktiven Systemen eine komplexe Herausforderung darstellt. Ziel dieser Arbeit ist es, Beiträge zum einfacheren Verständnis und zur Analyse des reaktiven Verhaltens von SPS-Programmen zu leisten.

1.1 Vorgehen

Diese Arbeit präsentiert einen Ansatz zur Analyse von SPS-Programmen, zur Verbesserung des Programmverständnisses und zum Aufspüren von Defekten.

In Kapitel 2 wird ein Verfahren zum Aufzeichnen und deterministischen Wiederabspielen von SPS-Programmen präsentiert. Dieses Verfahren bildet die Grundlage der vorliegenden Arbeit und liefert die Daten, auf deren Basis in den weiteren Kapiteln Programmanalysen durchgeführt werden.

In Kapitel 3 wird dann ein Formalismus zur Spezifikation des Verhaltens von reaktiven Systemen vorgestellt. Basierend auf dem Formalismus der *Hybriden Automaten* [Hen96] kann damit gewünschtes oder fehlerhaftes Verhalten des Systems spezifiziert werden. Weiters werden Algorithmen präsentiert, mit denen automatisch Spezifikationsverletzungen gefunden werden können.

Ein besonderes Problem der Analyse von SPS-Systemen ist, dass sich das Verhalten aus einer Interaktion zwischen dem Programmcode und dem damit verbundenen physikalischen System ergibt. Um diese Interaktion zu analysieren, wird in Kapitel 4 ein Verfahren zur Synthese eines abstraktes Modells eines SPS-Systems aufgestellt. Dieses Verhaltensmodell lässt sich als SMT-Formel codieren. So kann dieses Modell mit Model Checkern weiter verarbeitet werden, um zum Beispiel automatische Beweise über das Programmverhalten zu führen.

2 Replay Debugging

In diesem Kapitel wird ein Verfahren zur Aufzeichnung und zum deterministischen Wiederabspielen von SPS-Programmen vorgestellt. Dieses Verfahren wurde als Basis für die Analyse von SPS-Programmen entwickelt. Es liefert die Daten, die den in den weiteren Kapiteln vorgestellten Analyseverfahren zugrunde liegen.

Eine grundlegende Herausforderung in der Analyse von SPS-Programmen ist die Echtzeitfähigkeit. In der Literatur existieren verschiedene Ansätze zum deterministischen Wiederabspielen (z.B. [DKC+02, JO07, KDC05]). Die meisten dieser Ansätze sind entweder gar nicht für Echtzeit-Anwendungen geeignet, oder besitzen einen deutlich höheren Overhead als das hier vorgestellte Verfahren.

Abbildung 1 zeigt den generellen Ansatz des Verfahrens zum Aufzeichnen und Wiederabspielen von SPS-Programmen (siehe auch [PSWM11] und [Wir13]).

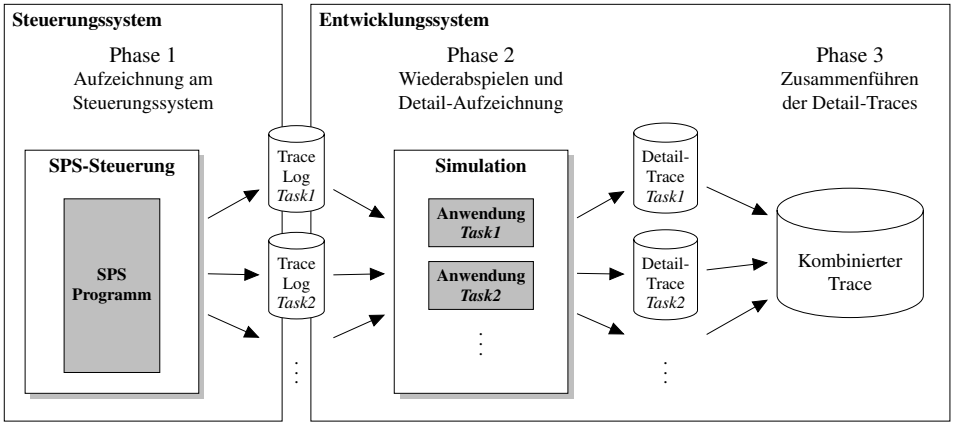


Abbildung 1: Mehrphasiges Verfahren zum deterministischen Wiederabspielen

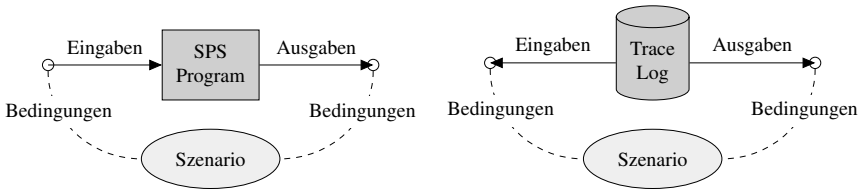
In der ersten Phase wird nur die Information aufgezeichnet, die zum Wiederabspielen eines einzelnen Tasks notwendig ist. Diese Aufzeichnung erfolgt für jeden nebenläufigen Task separat. In der zweiten Phase werden die Aufzeichnungen der ersten Phase verwendet, um jeden Task des SPS-Programms isoliert am Entwicklungssystem abzuspielen. Dabei werden zusätzliche Informationen über den Programmlauf aufgezeichnet. Dann werden diese Informationen zu einem kombinierten Trace-Log zusammengefasst, das auch Informationen über das Scheduling enthält.

Ein Vorteil dieses mehrphasigen Ansatzes ist, dass nur die erste Phase in einer Umgebung mit Echtzeitanforderungen laufen muss. In den späteren Phasen gibt es keine Ressourceneinschränkungen, es können beliebige Daten zur späteren Analyse gesammelt werden. Die folgenden Kapitel beschreiben Verfahren, die die so gesammelten Daten nutzen.

3 Modellbasiertes Testen

In Kapitel 2 wunde ein Verfahren zum deterministischen Wiederabspielen von Programmen vorgestellt. Im Gegensatz zu herkömmlichen Debugging-Techniken hat das deterministische Wiederabspielen den Vorteil, dass das Verhalten der Anwendung exakt reproduziert werden kann. Es löst alle Probleme mit der Reproduzierbarkeit von sporadischen Fehlern, allerdings ergibt sich eine neue Herausforderung: Wenn ein Entwickler die Aufzeichnung einer langlaufenden Anwendung bekommt, ist es sehr schwierig, einen bestimmten Fehler in der Aufzeichnung zu finden.

Dieses Kapitel präsentiert ein Verfahren, das modellbasiertes Testen eines reaktiven Programms erlaubt. Basieren auf Hybriden Automaten wird ein Formalismus entwickelt, der die Spezifikation von Ein-/Ausgabeverhalten eines reaktiven Systems erlaubt. Der Neuheitswert dieses Spezifikationsformalismus ist, dass die selbe Spezifikation sowohl zum Testen von Implementierungen verwendet werden kann, als auch zum Lokalisieren von



(a) Bedingungen auf E/A-Verhalten (b) Analyse von Trace basierend auf Szenario

Abbildung 2: Szenario-Spezifikation

Fehlern in einem aufgezeichneten Ereignisstrom. Der Entwickler kann ein Szenario spezifizieren, und auf Grund dieser Spezifikation eine Implementierung testen. Genauso können basierend auf der selben Spezifikation alle Stellen in einer Aufzeichnung gefunden werden, wo dieses Szenario auftritt.

3.1 Szenario-Spezifikation

Um Tests vollautomatisch ausführen zu können, ist ein Mechanismus nötig, mit dem das erwartete Verhalten spezifiziert werden kann. Bei SPS-Programmen bedeutet das, dass der Entwickler erwartetes oder ungültiges Verhalten des Systems in Bezug auf die Sensoren und Aktuatoren über die Zeit spezifizieren muss.

Durch deterministisches Wiederabspielen entfällt die Notwendigkeit, zur Reproduktion von Fehlern minimale Testfälle zu schreiben, jeder Fehler kann durch Wiederabspielen reproduziert werden. Allerdings muss der Fehler in einer langen Aufzeichnung lokalisiert werden. Um das zu automatisieren ist wieder ein Mechanismus zur Spezifikation des erwarteten Verhaltens notwendig. Ein Algorithmus kann dann die Aufzeichnung nach einer Stelle durchsuchen, an der das spezifizierte Szenario auftritt.

Abbildung 2 zeigt den generellen Ansatz. Szenarios werden als Bedingungen auf dem Ein-/Ausgabeverhalten spezifiziert. Ein Szenario definiert eine Menge von korrekten bzw. fehlerhaften Verhalten. Diese Szenario-Spezifikationen können zum Testen einer Implementierung verwendet werden: Ein Test-Treiber generiert gültige Eingaben aus der Szenario-Spezifikation, und überprüft ob die Ausgaben des Programms die Bedingungen aus der Spezifikation erfüllen (Abbildung 2a).

Die Aufzeichnungsanalyse funktioniert ähnlich: Die Ein- und Ausgabewerte werden aus dem Trace-Log gelesen, und es wird überprüft ob sie den Bedingungen in der Szenario-Spezifikation entsprechen (Abbildung 2b).

Das erwartete Verhalten eines reaktiven Programms und der physikalischen Umgebung kann mit dem Formalismus der *Hybriden Automaten* [Hen96] spezifiziert werden. Ein hybrider Automat modelliert das diskrete Verhalten eines hybriden Systems mit Zustandsübergängen in einem endlichen Automaten. Das stetige Verhalten des Systems wird mit Differentialgleichungen modelliert.

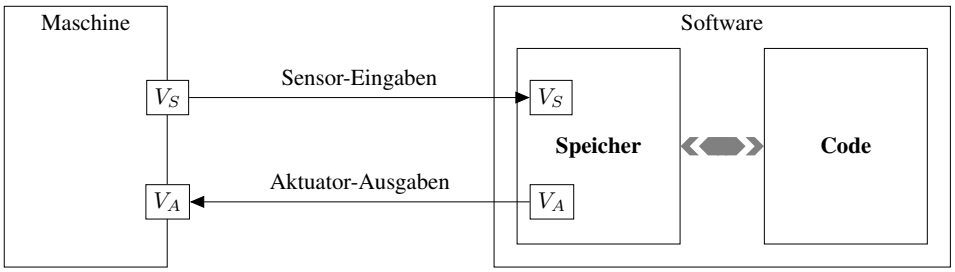


Abbildung 3: SPS-Programmausführung

Definition 1. *Ein Hybrider Automat ist ein Tupel* $H = (V, S, s_0, \text{init}, I, F, T)$.

V ist eine endliche Menge von Variablen. S ist eine endliche Menge von Zuständen, $s_0 \in S$ ist der Anfangszustand. init *ist die Anfangsbelegung der Variablen. $I(s)$ sind Invarianten über die Variablen, und $F(s)$ gibt Differentialgleichungen die die Entwicklung der Variablenwerte beschreiben. T ist die Übergangsrelation.*

Ein wichtiger Vorteil des gemeinsamen Spezifikationsformalismus für Trace-Analyse und Testen ist, dass nachdem eine Szenario-Spezifikation zum Auffinden eines Fehlers verwendet wurde, die selbe Spezifikation mit geringem Aufwand in einen Regressions-Test umgewandelt werden kann, der sicher stellt dass der selbe Fehler nicht später wieder auftritt.

3.2 Testausführung

Mit hybriden Automaten kann ein testbares Szenario spezifiziert werden, mit dem SPS-Programme vollautomatisch getestet werden können. Für eine rigorose Definition der Voraussetzungen an die Spezifikation, sowie Beweise der Korrektheit der Algorithmen, wird auf die Langfassung der Arbeit verwiesen [Sch13].

Abbildung 3 zeigt den typischen Aufbau eines Steuerungssystems: Die gesteuerte Maschine hat Sensoren (V_S) und Aktuatoren (V_A). Diese werden in den Speicher der Software abgebildet. Aus Sicht des Test-Treibers ist die Software-Box undurchsichtig, keine bestimmte Implementierung wird vorausgesetzt.

Die Implementierung eines Steuerungssystems wird im Test-Algorithmus durch einen Moore-Automaten abstrahiert. Der Test-Treiber simuliert die Maschine, indem der hybride Automat eines Testfalls parallel zum Moore-Automaten der Implementierung geschalten wird.

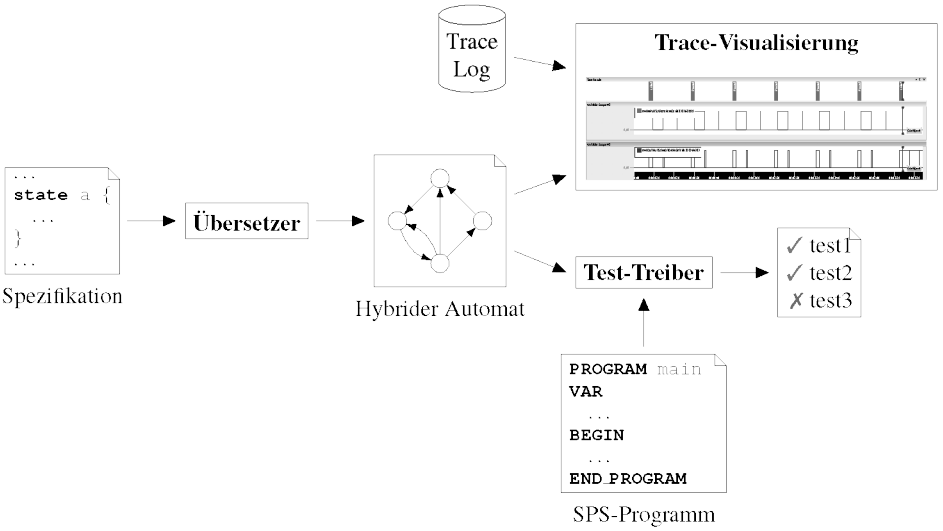


Abbildung 4: Test- und Trace-Analyse-Verfahren

3.3 Trace-Analyse

Mit dem selben Spezifikationsformalismus ist es auch möglich, eine Offline-Analyse eines aufgezeichneten Programmlaufs durchzuführen. Die Aufzeichnung von Programmläufen wurde in Kapitel 2 beschrieben (siehe auch [PSWM11]).

Der Algorithmus zur Trace-Analyse funktioniert ähnlich wie der Test-Algorithmus. Wieder wird der hybride Automat simuliert, anstatt allerdings die Werte von einem laufenden Programm zu lesen, werden alle Variablenwerte aus dem Trace-Log entnommen. So kann der Analysealgorithmus alle Sektionen im Trace-Log finden, die zu einer Szenariospezifikation passen, und auch korrektes von fehlerhaftem Verhalten unterscheiden.

Abbildung 4 gibt einen Überblick über das Test- und Trace-Analyse-Verfahren. Der Entwickler schreibt eine Spezifikation in einer Spezifikationssprache. Diese Spezifikation wird in einen hybriden Automaten übersetzt, der als Eingabe für den Test-Treiber zum modellbasierten Testen, oder zur Analyse eines Trace-Logs verwendet werden kann. Um die Spezifikation von Szenarios zu vereinfachen, wurde auch eine graphische Spezifikationssprache entwickelt.

4 Spezifikationssynthese

In Kapitel 2 wurde ein Verfahren präsentiert, um Programmläufe aufzuzeichnen und wiederabzuspielen. In Kapitel 3 wurde ein Verfahren präsentiert, mit dem in diesen Aufzeichnungen Stellen gesucht werden können, an denen Spezifikationen erfüllt oder verletzt wer-

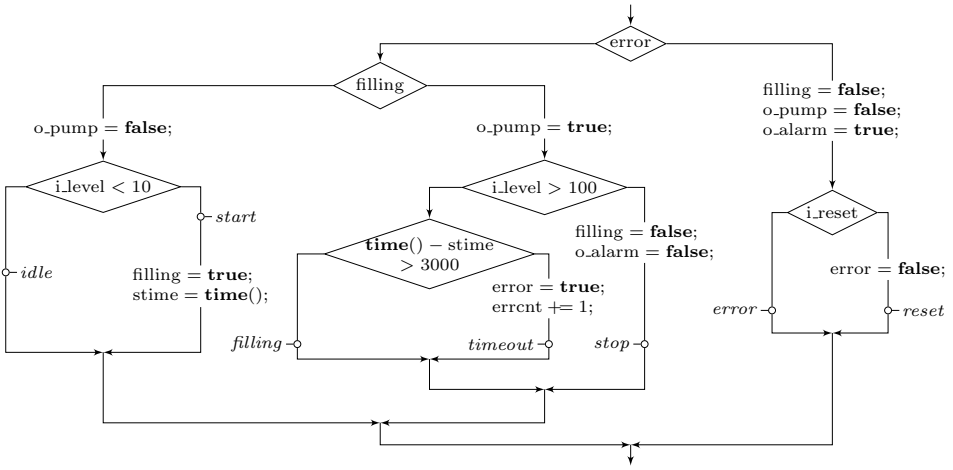


Abbildung 5: Programmcode einer Füllstandsregelung

den. In diesem Kapitel wird nun ein Verfahren vorgestellt, mit dem ein abstraktes Modell des Ein-/Ausgabeverhaltens eines SPS-Programms synthetisiert werden kann. Zuerst wird mittels symbolischer Ausführung des Programmcodes ein Automatenmodell gebaut. Mit Hilfe von Techniken aus dem Übersetzerbau wird daraus ein auf temporaler Logik basiertes Modell überführt. Dieses Modell kann als SMT-Formeln codiert werden, um so Model-Checking-Techniken darauf anzuwenden.

Der Neuheitswert des hier präsentierten Verfahrens ist, dass dieses Modell aus einer Kombination aus statischer Analyse des Quelltexts und dynamischer Information aus Programmaufzeichnungen erstellt. Dadurch werden die Vorteile von statischer und dynamischer Analyse kombiniert.

4.1 Reaktives Verhalten

Ein Steuerungssystem als Ganzes weist typischerweise komplexes Verhalten über die Zeit auf. Während das Verhalten der Software nur vom Quellcode bestimmt ist, ist das Verhalten des physikalischen Systems oft unbekannt und nur schwer vorhersagbar. Das Gesamtverhalten des Systems ist bestimmt von der komplexen Interaktion zwischen der Dynamik des physikalischen Geräts und der Software, und dieses komplexe Zusammenspiel kann nicht durch eine Analyse der Software alleine erfasst werden.

Um dieses Problem zu lösen ist es Zielführend, die formale Analyse des Programmcodes mit Beobachtungen der Reaktionen des physikalischen Geräts zu kombinieren. Die Beobachtungen können mit dem Aufzeichnungs- und Wiederabspiel-Verfahren von Kapitel 2 erzeugt werden.

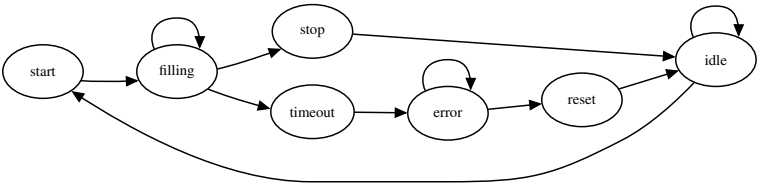


Abbildung 6: Transitionsgraph der Füllstandsregelung

Abbildung 5 zeigt den Programmcode einer einfachen Füllstandsregelung: Eine Pumpe muss so gesteuert werden, dass der Füllstand immer zwischen zwei Grenzwerten gehalten wird.

Von diesem Programm kann ein Transitionsmodell des Kontrollflusses erzeugt werden, indem eine Aufzeichnung eines Programmlaufs wiederabgespielt wird, wobei in jedem Zweig des Programms eine Trace-Anweisung eingefügt wird. Abbildung 6 zeigt den so gewonnenen Transitionsgraphen der Füllstandsregelung.

Mit Hilfe symbolischer Ausführung der einzelnen Kontrollflusspfade des Programms kann dieser Transitionsgraph nun mit Informationen aus dem Quelltext des Programms ergänzt werden, um ein vollständiges Modell des reaktiven Verhaltens des SPS-Programms zu bauen.

Definition 2. *Ein* Reaktives Programm *ist ein Tupel* $(V_i, V_e, C, \rightsquigarrow, P, U)$.

V_i und V_e sind die internen bzw. externen Variablen. C ist eine Menge von Kontrollflusspfaden, $\rightsquigarrow \subset C \times C$ ist die Transitionsrelation.

$P(c)$ ist eine Formel, die die Verzweigungsbedingungen des Kontrollflusspfads $c \in C$ beschreibt, und $U(c, v_i)$ gibt eine Update-Formel für die interne Variable v_i im Kontrollflusspfad c.

4.2 Verhaltensmodell

Das im letzten Abschnitt konstruierte Modell des Programmverhaltens enthält Formeln, die das Verhalten des Programms vollständig beschreiben. In diesem Abschnitt wird dieses Modell benutzt, um Wissen über das extern sichtbare Ein- und Ausgabeverhalten eines SPS-Programms herzuleiten. Dieses Wissen wird als past-time temporale Ausdrücke über die Sensor- und Aktuator-Werte formuliert:

$$\mathcal{T} \quad ::= \quad v_e \mid \text{‘}\mathbf{h}\text{’ ‘(’ } c \text{ ‘)’} \mid \text{‘}\mathbf{Y}\text{’ } \mathcal{T} \mid \text{‘}\mathbf{D}\text{’ } \mathcal{T}$$

$\mathbf{Y}$ ist der „vorher“-Operator, der auf einen Wert im vorhergegangenen Zeitschritt verweist. $\mathbf{h}(c)$ gibt an, dass im aktuellen Zeitschritt der Pfad c ausgeführt wurde.

Zu diesen Operatoren kommt noch $\mathbf{D}$, der „Dominator“-Operator: Er verweist auf Werte, die bei der letzten Ausführung des Dominator-Pfades aufgetreten sind:

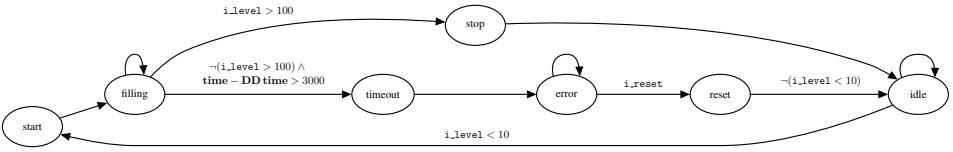


Abbildung 7: Übergangsbedingungen

Definition 3. *Ein Pfad c* dominiert *einen Pfad c' (kurz: $c \gg c'$), wenn jeder gültige Programmlauf $c_0 \rightsquigarrow c_1 \rightsquigarrow \cdots \rightsquigarrow c'$ den Pfad c enthält [CFR+91].*

Der Vorteil dieses Operators ist, dass der Dominator einen Referenzpunkt in der Vergangenheit bietet, der immer existiert, und damit Wissen generiert werden kann, das unbedingt gilt. Mit Hilfe eines aus dem Übersetzerbau stammenden Algorithmus [CFR+91] kann nun das abstrakte Verhaltensmodell in ein temporales Prädikat G umgewandelt werden, das gültige Programmläufe charakterisiert.

Das Prädikat G kann in eine Form gebracht werden, die von einem SMT-Solver gelesen werden kann. Der SMT-Solver kann dann dazu verwendet werden, *Übergangsbedingungen* abzuleiten. Diese Bedingungen beschreiben den Unterschied zwischen zwei Zuständen im Transitionsgraph anhand der Änderung der Eingabewerte.

Abbildung 7 zeigt den Transitionsgraphen des Füllstandsregler-Beispiels mit den automatisch generierten Übergangsbedingungen. Dieses Beispiel illustriert einige Vorteile dieses Verfahrens zum Programmverständnis. Zum Beispiel ist der Übergang $\mathit{filling} \rightsquigarrow \mathit{stop}$ mit `i_level` > 100 beschriftet. *filling* ist also ein Wartezustand, in dem darauf gewartet wird, dass der Füllstand 100 erreicht.

5 Zusammenfassung

Diese Arbeit beschreibt Methoden zum Testen und zur Analyse des reaktiven Verhaltens von SPS-Programmen. Die Hauptbeiträge dieser Arbeit sind: (1) Ein Verfahren wird vorgestellt, das die Aufzeichnung und das deterministische Wiederabspielen von SPS-Programmen ermöglicht. (2) Es wird ein Spezifikationsformalismus vorgestellt, der basierend auf hybriden Automaten das modellbasierte Testen von SPS-Programmen ermöglicht. (3) Es wird ein Verfahren präsentiert, das mit einer Kombination aus Model-Checking-Techniken und Techniken aus dem Übersetzerbau vollautomatisch ein Modell des Ein-/Ausgabeverhaltens von SPS-Programmen in Form von temporaler Logik synthetisiert.

Die in dieser Arbeit vorgestellten Verfahren wurden anhand von Beispielen aus der Literatur und anhand einer Fallstudie mit einem realistischen Steuerungsprogramm unseres Industriepartners evaluiert.

Literatur

[CFR+91] Ron Cytron, Jeanne Ferrante, Barry K. Rosen, Mark N. Wegman und F. Kenneth Zadeck. Efficiently Computing Static Single Assignment Form and the Control Dependence Graph. *ACM Trans. Program. Lang. Syst.*, 13(4), 1991.

[DKC+02] George W. Dunlap, Samuel T. King, Sukru Cinar, Murtaza A. Basrai und Peter M. Chen. ReVirt: Enabling Intrusion Analysis Through Virtual-Machine Logging and Replay. In *OSDI*, 2002.

[Hen96] Thomas A. Henzinger. The Theory of Hybrid Automata. In *LICS*, 1996.

[HP85] David Harel und Amir Pnueli. Logics and models of concurrent systems. Kapitel On the development of reactive systems. Springer-Verlag New York, Inc., New York, NY, USA, 1985.

[JO07] Shrinivas Joshi und Alessandro Orso. SCARPE: A Technique and Tool for Selective Capture and Replay of Program Executions. In *ICSM*, 2007.

[KDC05] Samuel T. King, George W. Dunlap und Peter M. Chen. Debugging Operating Systems with Time-Traveling Virtual Machines. In *USENIX Annual Technical Conference, General Track*, 2005.

[PSWM11] Herbert Prähofer, Roland Schatz, Christian Wirth und Hanspeter Mössenböck. A Comprehensive Solution for Deterministic Replay Debugging of SoftPLC Applications. *IEEE Trans. Industrial Informatics*, 7(4), 2011.

[Sch13] Roland Schatz. *Trace-based Testing of Multi-threaded Reactive Systems*. Dissertation, Johannes Kepler University, Linz, Austria, 7 2013.

[Wir13] Christian Wirth. *Dynamic Analysis of Multi-threaded Real-time PLC Applications using Record and Replay*. Dissertation, Johannes Kepler University, Linz, Austria, 6 2013.

Roland Schatz wurde geboren am 2. Oktober 1982. Nach Volks- und Hauptschule in Naarn absolvierte er die HTL für EDV und Organisation in Leonding, wo er 2002 die Matura mit Auszeichnung bestand. Anschließend absolvierte er in Linz ein Bachelor- und Diplomstudium der Informatik. In dieser Zeit nahm er an internationalen Programmierwettbewerben teil, und erreichte 2007 im Welt-Finale des ACM International Collegiate Programming Contest in Tokio den 26. Platz. Ab 2006 arbeitete er als Forscher im Christian Doppler Labor für Automated Software Engineering, wo er auch 2013 seine Dissertation im Projekt „Testing and Diagnosis of Programmable Logic Controller Programs“ verfasste. Am 2. Juni 2014 wurde im der Doktortitel unter den Auspizien des Bundespräsidenten verliehen. Seit 2013 arbeitet er als Forscher bei Oracle Labs Austria im Projekt „Graal – a next generation compiler for the HotSpot VM“.