

[dısi: arti:]

ANDREAS WINDISCH

SUCHBASIERTER STRUKTURTEST FÜR SIMULINK
MODELLE

SUCHBASIERTER STRUKTURTEST FÜR SIMULINK MODELLE

vorgelegt von
Dipl.-Inform. Andreas Windisch
aus Berlin

Von der Fakultät IV – Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
— Dr.-Ing. —

genehmigte Dissertation

PROMOTIONS AUSSCHUSS

Prof. Dr. rer. nat. Peter Pepper (Vorsitzender)
Prof. Dr.-Ing. Stefan Jähnichen (Gutachter)
Prof. Dr. rer. nat. Helmuth A. Partsch (Gutachter)

TAG DER WISSENSCHAFTLICHEN AUSSPRACHE:
29. September 2011

Berlin 2011
D 83

Technische Universität Berlin

Fachgebiet Softwaretechnik

Sekretariat TEL 12-3

Ernst-Reuter-Platz 7

D-10587 Berlin

Universität Ulm

Institut für Programmiermethodik und Compilerbau

James-Franck-Ring, Gebäudekreuz 027

D-89069 Ulm

Andreas Windisch: *Suchbasierter Strukturtest für Simulink Modelle*

Für meine Frau
Isabelle-Désirée Windisch

KURZFASSUNG

Die strukturorientierte Testdatengenerierung ist eine besonders aufwändige Aufgabe beim Test komplexer Modelle. Eine Automatisierung dieser Aufgabe verspricht signifikante Kosteneinsparungen und eine Erhöhung der erzielten Testqualität. Existierende Ansätze zur Automatisierung des strukturorientierten Tests für Modelle basieren hauptsächlich auf symbolischer Ausführung und Constraint Solving oder der Zufallssuche. Diese Ansätze unterliegen jedoch insbesondere für komplexe Modelle Einschränkungen, die zu einer starken Begrenzung sowohl ihrer Anwendbarkeit als auch der von ihnen erzielten strukturellen Überdeckung des Modells führen. Darüber hinaus spielt die Plausibilität der generierten Testdaten bei den existierenden Ansätzen eine untergeordnete Rolle.

In dieser Arbeit wird ein Verfahren zur Automatisierung der strukturorientierten Testdatengenerierung für komplexe Modelle entwickelt. Die Aufgabe der strukturorientierten Testdatengenerierung wird in ein Optimierungsproblem transformiert, welches mit Hilfe eines metaheuristischen Suchverfahrens gelöst wird. Während der Suche werden Anforderungen des Testers an die zu erstellenden Testdaten berücksichtigt. Diese Anforderungen umfassen einfache Amplitudenbeschränkungen ebenso wie komplexe Signal Constraints.

Das Ergebnis ist ein automatisiertes Verfahren, welches die Generierung plausibler Testdaten zur Erreichung einer hohen strukturellen Überdeckung für komplexe Modelle ermöglicht. Die Effektivität des Verfahrens konnte anhand einer experimentellen Fallstudie nachgewiesen werden. Im Vergleich mit existierenden Verfahren konnte eine signifikant höhere strukturelle Überdeckung der verwendeten Testobjekte erreicht werden. Ferner wurden die spezifizierten Signal Constraints nahezu vollständig erfüllt.

ABSTRACT

Structure-oriented test data generation is a particularly laborious task in testing complex models. An automation of this task promises significant cost reductions and an improvement of the realized test quality. Existing approaches for automating structure-oriented testing of models are primarily based on symbolic execution and constraint solving or random search respectively. In particular for complex models these approaches are subject to restrictions, which lead to limitations of both their applicability and the achieved structural coverage of the model. In addition, plausibility of generated test data plays a less significant role in existing approaches.

In this work a new approach for automating structure-oriented test data generation for complex models is developed. The task of generating suitable test data is transformed in an optimization problem, which is attempted to be solved by means of a meta-heuristic search technique. Specific requirements of the tester on the test data to be generated are considered during search. These requirements include simple amplitude restrictions as well as more complex signal constraints.

The outcome of this work is an automated approach allowing for the generation of plausible test data for reaching high structural coverage for complex models. The effectiveness of the approach has been demonstrated using an experimental case study. In comparison to existing approaches a significantly higher structural coverage of the used test objects has been reached. Furthermore, the specified signal constraints have almost completely been achieved.

VERÖFFENTLICHUNGEN

Die in dieser Dissertation vorgestellte Arbeit entstand zwischen Mai 2007 und Juni 2011 an der Technischen Universität Berlin (Daimler Center for Automotive Information Technology Innovations). Einige Ideen und Darstellungen sind bereits in den im Folgenden aufgelisteten vorherigen Veröffentlichungen erschienen:

KONFERENZBEITRÄGE:

- F. Lindlar, **A. Windisch**: *A Search-Based Approach to Functional Hardware-in-the-Loop Testing*, 2nd International Symposium on Search-Based Software Engineering (SSBSE 2010), September 2010, Benevento, Italien
- **A. Windisch**: *Search-Based Test Data Generation from Stateflow State-charts*, 12th ACM Genetic and Evolutionary Computation Conference (GECCO 2010), Juli 2010, Portland, USA
- A. Baars, P. Kruse, F. Lindlar, T. Vos, J. Wegener und **A. Windisch**: *Industrial Scaled Automated Structural Testing with the Evolutionary Testing Tool*, 3rd IEEE International Conference on Software Testing, Verification and Validation (ICST 2010), April 2010, Paris, Frankreich
- B. Wilmes und **A. Windisch**: *Considering Signal Constraints in Search-Based Testing of Continuous Systems*, 3rd International Search-Based Software Testing Workshop in conjunction with 3rd IEEE International Conference on Software Testing, Verification and Validation (ICST 2010), April 2010, Paris, Frankreich
- F. Lindlar, **A. Windisch** und J. Wegener: *Integrating Model-Based Testing with Evolutionary Functional Testing*, 3rd International Search-Based Software Testing Workshop in conjunction with 3rd IEEE International Conference on Software

Testing, Verification and Validation (ICST 2010), April 2010, Paris, Frankreich

- **A. Windisch**, F. Lindlar, S. Topuz und S. Wappler: *Evolutionary Functional Testing of Continuous Control Systems*, 11th ACM Genetic and Evolutionary Computation Conference (GECCO 2009), Juli 2009, Montreal, Kanada
- **A. Windisch** und N. Al Moubayed: *Signal Generation for Search-Based Testing of Continuous Systems*, 2nd International Search-Based Software Testing Workshop in conjunction with 2nd IEEE International Conference on Software Testing, Verification and Validation (ICST 2009), April 2009, Denver, USA
- **A. Windisch**: *Ideas on Signal Generation for Evolutionary Testing of Continuous Systems*, Dagstuhl Seminar Proceedings: Evolutionary Test Generation, 2009, Dagstuhl, Germany
- **A. Windisch**: *Search-Based Testing of Complex Simulink Models containing Stateflow Diagrams*, 1st Search-Based Software Testing Workshop in conjunction with 1st IEEE International Conference on Software Testing, Verification and Validation (ICST 2008), April 2008, Lillehammer, Norwegen

DOKTORANDENSYMPOSIEN:

- **A. Windisch**: *Search-Based Testing of Complex Simulink Models containing Stateflow Diagrams*, 31st ACM/IEEE International Conference on Software Engineering (ICSE 2009), Mai 2009, Vancouver, Kanada
- **A. Windisch**: *Search-Based Testing of Complex Simulink Models containing Stateflow Diagrams*, 2nd IEEE International Conference on Software Testing, Verification and Validation (ICST 2009), April 2009, Denver, USA

INHALTSVERZEICHNIS

I	GEGENSTANDSBEREICH	1
1	EINFÜHRUNG	3
1.1	Ziele und Lösungsansatz	6
1.2	Beiträge	8
1.3	Struktur	9
2	GRUNDLAGEN UND VERWANDTE ARBEITEN	11
2.1	Modell-Strukturtest	11
2.1.1	Simulationsmodelle	12
2.1.2	MATLAB/Simulink/Stateflow	13
2.1.3	Softwaretest allgemein	16
2.1.4	Modelltest	18
2.1.5	Strukturorientiertes Testen	19
2.2	Automatische Testdatengenerierung	23
2.2.1	Statische Testdatengenerierung	24
2.2.2	Dynamische Testdatengenerierung	31
2.2.3	Kommerzielle Werkzeuge	42
2.2.4	Grenzen existierender Ansätze	44
2.3	Evolutionäre Algorithmen	48
2.3.1	Prinzipien evolutionärer Algorithmen	49
2.3.2	Genetische Programmierung	57
2.4	Zusammenfassung	63
II	SUCHBASIERT. STRUKTURTEST VON SIMULINK MODELLEN	65
3	SIGNALGENERIERUNG UND -OPTIMIERUNG	67
3.1	Ansätze	67
3.1.1	Trigonometrische Polynome	69
3.1.2	Segmentbasierter Ansatz	72
3.1.3	Längenvariabler segmentbasierter Ansatz	79
3.1.4	Empirischer Vergleich	81
3.2	Bewertung von Signalen	88
3.3	Signal-Constraints	91
3.3.1	Temporale Logik für Signale	93
3.3.2	Berücksichtigung während der Suche	99
3.3.3	Distanzmaße	104
3.4	Zusammenfassung	122

4	ÜBERDECKUNG VON SIMULINK/STATEFLOW	125
4.1	Überdeckungskriterien	125
4.2	Instrumentierung	128
4.2.1	Simulink Blöcke	129
4.2.2	Stateflow Diagramme	130
4.3	Distanzmetriken	133
4.3.1	Simulink Blöcke	133
4.3.2	Stateflow Diagramme	138
4.4	Zusammenfassung	146
5	FALLSTUDIE	149
5.1	Realisierung des Verfahrens	150
5.2	Aufbau der Fallstudie	153
5.2.1	Forschungsfrage	153
5.2.2	Testobjektauswahl	154
5.2.3	Versuchsaufbau	159
5.2.4	Geplante Datenerhebung	163
5.2.5	Kriterien zur Ergebnisinterpretation	165
5.2.6	Bedrohung der Fallstudienvolidität	165
5.3	Durchführung der Fallstudie	167
5.3.1	Datenerhebung	167
5.3.2	Bewertung	172
5.4	Zusammenfassung	175
III	ABSCHLUSS	177
6	ZUSAMMENFASSUNG UND AUSBLICK	179
6.1	Ergebnisse dieser Arbeit	179
6.2	Einschränkungen und Grenzen	181
6.3	Weiterführende Arbeiten	182
IV	ANHANG	187
A	ANHANG	189
A.1	Parameter Tuning	189
A.2	Überdeckungsziele	190
A.3	Fallstudienergebnisse	196
	LITERATURVERZEICHNIS	199

ABBILDUNGSVERZEICHNIS

Abb. 1	Übersicht über den entwickelten Ansatz	8
Abb. 2	Signaldiskretisierung	13
Abb. 3	Beispielhaftes Simulink Modell	15
Abb. 4	Beispielhaftes Stateflow Diagramm	16
Abb. 5	Beispielprogramm und CFG	21
Abb. 6	Bedingungsüberdeckung für SL und SF	23
Abb. 7	Symbolische Ausführung: Beispiel	25
Abb. 8	Optimierungsprozess eines Teilziels	33
Abb. 9	Beispielhafte Kontrollflussdarstellungen	35
Abb. 10	Instrumentierung nach Zhan	39
Abb. 11	Zielfunktionsberechnung nach Lefticaru/Ipate . .	41
Abb. 12	Anwendung eines evolutionären Algorithmus' . .	51
Abb. 13	Ablauf eines evolutionären Algorithmus'	52
Abb. 14	Stochastic Universal Sampling	54
Abb. 15	Turnierselektion	55
Abb. 16	Kodierung für LGP	59
Abb. 17	LGP Rekombinationsoperatoren	60
Abb. 18	LGP Mutationsoperatoren	62
Abb. 19	Beispiel eines zeitdiskreten Signals	68
Abb. 20	Grundlegendes Prinzip des Signalgenerators . .	69
Abb. 21	Periodizität trigonometrischer Polynome	72
Abb. 22	Signalaufbau beim segmentbasierten Ansatz . .	73
Abb. 23	Basissignale des segmentbasierten Ansatzes . .	74
Abb. 24	Verknüpfung von Signalsegmenten	77
Abb. 25	Segmentbasierte Signalkonstruktion	78
Abb. 26	Genotypische Kodierung für den LvSb-Ansatz . .	80
Abb. 27	Ergebnis der experimentellen Fallstudie	86
Abb. 28	Effektivitätsvergleich der Verfahren	88
Abb. 29	Berechnung eines einzelnen Zielfunktionswertes .	89
Abb. 30	Syntax eines Signal-Constraints	95
Abb. 31	Semantische Auswert. eines Signal Constraints . .	96
Abb. 32	Einbettung der Penalty Funktion	102
Abb. 33	Partitionierung des Suchraums	105
Abb. 34	Distanzberechn. für Vergleichsoperatoren	110
Abb. 35	Distanzberechn. für aussagenlog. Operatoren . .	112

Abb. 36	Distanzberechn. für temporale Operatoren	116
Abb. 37	Distanzberechn. für Ereignis-Operatoren (1/2) . .	120
Abb. 38	Distanzberechn. für Ereignis-Operatoren (2/2) . .	121
Abb. 39	Notwendige Beobachtungen für SL/SF Objekte .	128
Abb. 40	Instrumentierung von Simulink Blöcken	130
Abb. 41	Instrumentierung von Stateflow Diagrammen . .	131
Abb. 42	Distanzberechnung für Simulink Blöcke (1/3) . .	134
Abb. 43	Distanzberechnung für Simulink Blöcke (2/3) . .	136
Abb. 44	Distanzberechnung für Simulink Blöcke (3/3) . .	138
Abb. 45	Übersicht der Stateflow Distanzmetriken	139
Abb. 46	SBGen-SL: Systemarchitektur	151
Abb. 47	Transformation von Integrator Blöcken	157
Abb. 48	Transformation von State Space Blöcken	158
Abb. 49	Transformation von Fcn Blöcken	159
Abb. 50	Prozent. Erreichbarkeit der Überdeckungsziele . .	168
Abb. 51	Erreichte Überdeckung für die vier Testobjekte . .	173
Abb. 52	Gemittelte Ergebnisse der Fallstudie	174
Abb. A.1	SBGen-SL: Parameterbestimmung	189
Abb. A.2	Erreichung der Überdeckungsziele von TO1 . . .	196
Abb. A.3	Erreichung der Überdeckungsziele von TO2 . . .	197
Abb. A.4	Erreichung der Überdeckungsziele von TO3 . . .	197

TABELLENVERZEICHNIS

Tab. 1	Spezifische Distanzfunktionen	37
Tab. 2	Konfiguration des LvSb-Ansatzes	83
Tab. 3	Distanzmaße der Vergleichsoperatoren	109
Tab. 4	Distanzmaße der aussagenlogischen Operatoren .	111
Tab. 5	Distanzmaße δ^e der temporalen Operatoren . . .	113
Tab. 6	Distanzmaße δ^v der temporalen Operatoren . . .	114
Tab. 7	Distanzmaße der Ereignis-Operatoren	118
Tab. 8	Für DC, CC, MC/DC relevante SL/SF Elemente .	127
Tab. 9	Eigenschaften der ausgewählten Testobjekte . . .	156
Tab. 10	Konfiguration von SBGen-SL	161
Tab. 11	Konfiguration von Reactis Tester	162
Tab. 12	Testspezifikation für die Fallstudie	163
Tab. 13	Fallstudienergebnisse: Zufallstest	169

Tab. 14	Fallstudienergebnisse: Reactis Tester	170
Tab. 15	Fallstudienergebnisse: SBGen-SL	171
Tab. A.1	Überdeckungsziele von Testobjekt TO1	192
Tab. A.2	Überdeckungsziele von Testobjekt TO2	195
Tab. A.3	Überdeckungsziele von Testobjekt TO3	195

Teil I

GEGENSTANDSBEREICH

1 | EINFÜHRUNG

*There are two ways
to write error-free programs;
only the third one works.*

— Alan Perlis

Moderne Datenverarbeitungssysteme übernehmen zunehmend komplexere und verantwortungsvollere Aufgaben, sodass ein mögliches Fehlverhalten zu teils katastrophalen Auswirkungen führen kann, wie beispielsweise in den Bereichen der Kern- und Verkehrstechnik. Insbesondere gilt dies für eingebettete Systeme, welche mit minimalen Systemressourcen oft kritische Anwendungen in Echtzeit bearbeiten müssen und dabei auf maximale Zuverlässigkeit ausgelegt sind. Oft ist nicht mehr die Hardware sondern vielmehr die Software ein limitierender Faktor für Neuerungen [14].

*Relevanz
von
Software*

Besonders bemerkbar macht sich dieser Trend in der Automobilindustrie: Schätzungsweise 90% der Funktionalität in automobilen Systemen basiert auf Software [43]. Software wird in diesem Bereich als die wichtigste Technologie für Produktneuerungen und -innovationen angesehen [23] und trägt dadurch bedeutend zur gesamten Wertschöpfung des Fahrzeugs bei [114]. Die Software leistet zudem einen erheblichen Beitrag zur Produktdifferenzierung und wird daher zunehmend zu einem wettbewerbsentscheidenden Faktor [43]. Software realisiert zunehmend höchst sicherheitsrelevante Systeme, deren Ausfall bzw. Fehlverhalten gefährliche Auswirkungen haben kann. Beispiele für derartige Systeme sind sowohl Spurhalteassistenten, welche aktiv in die Lenkung des Fahrzeugs eingreifen, als auch moderne Abstandsregeltempomaten, welche einen direkten adaptierenden Einfluss auf die Fahrzeuggeschwindigkeit haben.

*Software in
der Automobi-
lindustrie*

Die Qualitätssicherung für die Software eingebetteter Systeme gewinnt daher immer mehr an Bedeutung, wobei die Fehlervermeidung und -erkennung bereits während möglichst früher

*Qualitäts-
sicherung*

*Modellbas.
Entwickl.*

Stufen des Entwicklungsprozesses eine der größten Herausforderungen der modernen Softwareentwicklung darstellt. Aus diesem Grund und aufgrund der zunehmenden Komplexität der entwickelten Systeme findet seit Ende der 1990er Jahre ein Paradigmenwechsel von der klassischen Softwareentwicklung zur modellbasierten Entwicklung statt [30, 140]. Während die klassische Softwareentwicklung durch textuelle Spezifikationen und manuelle Kodierung gekennzeichnet ist, wird die modellbasierte Entwicklung durch simulierbare Modelle und automatische Kodierung charakterisiert. So steht bereits in einem frühen Stadium des Entwicklungsprozesses unter Verwendung von Blockschaltbildern und Zustandsübergangsdiagrammen ein ausführbares Modell des zu entwickelnden Systems zur Verfügung, welches frühzeitig analysiert und getestet werden kann (Rapid Prototyping). Es kommen zunehmend grafische Programmiersprachen (Modellierungswerkzeuge) zum Einsatz [62], wie beispielsweise MATLAB® (ML)/Simulink® (SL)/Stateflow® (SF) [126, 128, 129], LabVIEW [99] oder die ASCET Produktfamilie [39]. Die zuerst genannten Modellierungs- und Simulationswerkzeuge (ML/SL/SF) haben sich vor allem in der Automobilindustrie zu einem De-facto-Standard entwickelt [31, 116, 119]. Bei der Daimler AG werden schätzungsweise bereits 90% aller Softwareneuentwicklungen eingebetteter Systeme mit ML/SL/SF entwickelt.

Softwaretest

Das Testen von Software ist eine der wichtigsten Maßnahmen zur Erhöhung des Vertrauens in die Korrektheit und Zuverlässigkeit von Software [35]. Durchschnittlich die Hälfte der Entwicklungskosten für Software wird zu Testzwecken verwendet [15, 52, 69, 123]; diese Abschätzung wurde durch eine im Jahr 2000 durchgeführte Studie bei der DaimlerChrysler AG bestätigt [55]. Eine besonders kritische Aktivität beim Softwaretest ist die Auswahl der Testdaten. Für einen Beweis der Korrektheit der Software müsste ein erschöpfender Test durchgeführt werden, d.h. das Testobjekt (das zu testende System) müsste mit allen möglichen Eingabewerten und allen möglichen Kombinationen ausgeführt werden. Da dies jedoch in der Praxis nicht durchführbar ist, muss ein Stichprobentest durchgeführt werden, d.h. für das jeweilige Testobjekt müssen fehlerrelevante Eingabedaten (Testdaten) bestimmt werden.

Automatisierung

Die Aufgabe der Testdatenermittlung ist bei manueller Durchführung besonders aufwändig und kostenintensiv. Eine Automatisierung ist daher besonders für komplexe Testobjekte zwingend

notwendig. Zur Automatisierung der Testdatenermittlung kommen verschiedene Techniken zum Einsatz. Eine dieser Techniken ist das strukturorientierte Testen (Strukturtest). Ein solches strukturorientiertes Testdatenermittlungsverfahren verwendet die Implementierung des Testobjekts, um die Aktivität der Testdatenermittlung zu steuern und die Relevanz gegebener Testdaten zu beurteilen. Strukturtestverfahren wenden zur Ermittlung relevanter Testdaten strukturelle Überdeckungskriterien an. Eines dieser Kriterien ist die Zweigüberdeckung, welche von den Testdaten die mindestens einmalige Ausführung eines jeden Zweiges des Kontrollflussgraphen des Testobjekts fordert. Die Erfüllung unterschiedlicher struktureller Überdeckungskriterien wird von einer Vielzahl von Standards für die Entwicklung sicherheitsrelevanter Software gefordert bzw. empfohlen: *DO-178B* Standard in der Luftfahrt [110], die *Norm 00-55* in der Rüstungsindustrie [139] und die entstehende ISO Norm 26262 im Automobilbereich [71, 72].

*Struktur-
tests*

Strukturorientierte Testdatengenerierungsverfahren wurden ursprünglich für den Test prozeduraler Software entwickelt. Diese Verfahren werden in den letzten Jahren zunehmend für den Test von Modellen angepasst (Modelltest). Es existieren Ansätze, welche die strukturorientierte Erstellung von Testdaten für Modelle automatisieren, um dadurch besonderen Nutzen aus der frühzeitigen Ausführbarkeit der Modelle im Rahmen der modellbasierten Entwicklung zu ziehen. Sie analysieren zu diesem Zweck die interne Struktur des zu testenden Modells und lassen sich grob in statische und dynamische Verfahren unterteilen. Statische Verfahren basieren u.a. auf symbolischer Ausführung und dem Lösen von Constraints (engl. Constraint Solving) [27, 67, 75], während dynamische Verfahren auf der tatsächlichen Ausführung des Testobjekts mit konkreten Eingabedaten und der Anwendung einer Suchstrategie basieren [85]. Die gemeinsame Herangehensweise besteht in der Unterteilung des Tests in Überdeckungsziele, basierend auf dem gewählten Strukturüberdeckungskriterium. Beispielsweise entspricht jeder zu erreichende Zweig des Kontrollflussgraphen des Testobjekts für die Zweigüberdeckung einem Überdeckungsziel, welches es zu erreichen gilt; für jedes Teilziel wird ein separater Versuch unternommen, einen Testfall zu erstellen.

*Struktur-
test-
verfahren*

Die existierenden Ansätze unterliegen allerdings Einschränkungen, wodurch ihre Anwendbarkeit für Modelle stark begrenzt wird: Bei den statischen Verfahren steigt bei besonders komplexen

*Grenzen
exist.
Ansätze*

Testobjekten die Anzahl der symbolischen Zustände mit steigender Komplexität des Testobjekts überproportional an (engl. state space explosion) [141]. Die Skalierbarkeit für komplexe Testobjekte ist somit nicht gegeben. Ein Beispiel dafür ist die Verarbeitung von Schleifen, welche zu einer potentiell unendlichen Menge symbolischer Zustände führen kann. Darüber hinaus stoßen heutige Constraint Solving-Techniken bei nicht-linearen und komplexen Constraints an ihre Grenzen [108]. Ein erfolgreiches Lösen der Constraints zur Bestimmung zielführender Eingabedaten ist aus diesem Grund nicht möglich. Die Einschränkungen der dynamischen Ansätze beziehen sich hauptsächlich auf die Effektivität und Effizienz der angewandten Suchstrategie. Die Suche nach einem Testdatum, welches auf einer besonders komplexen Bedingung basiert, kann aufgrund des Entstehens lokaler Optima erfolglos verlaufen. Die Effizienz der Verfahren wird häufig durch Generierung von nicht ausführbaren Lösungen verringert; die häufige Notwendigkeit der manuellen Spezifikation von detaillierten Informationen verringert zudem die Effektivität vorhandener dynamischer Ansätze. Keiner der bisherigen strukturorientierten Ansätze – sowohl statisch als auch dynamisch – berücksichtigt in angemessenem Umfang die Notwendigkeit, plausible Systemeingaben zu generieren. Ebenso werden möglicherweise notwendige Signal Constraints, wie beispielsweise Abhängigkeiten zwischen unterschiedlichen Systemeingaben, nicht berücksichtigt und führen so zu Testdaten, die für einen sinnvollen Test des Modells nicht verwendet werden können.

1.1 ZIELE UND LÖSUNGSANSATZ

*Hauptziel
der Arbeit*

Diese Arbeit beschreibt einen neuartigen, strukturorientierten Ansatz zur automatisierten Generierung von Testdaten für simulierbare Modelle (Suchbasierte Testdatengenerierung für Simulink – *SBGen-SL*). Das hauptsächliche Ziel besteht darin, die Einschränkungen der existierenden Automatisierungsansätze aufzuheben, um dadurch die Effektivität und Effizienz des Modell-Strukturtests zu erhöhen und diesen dadurch auch im industriellen Umfeld anwendbar zu machen:

- A. Grenzen der symbolischen Ausführung und des Constraint Solvings,

- B. Schwächen beim Vorhandensein komplexer Prädikate,
- C. Fehlende Beachtung von Zustandsdiagrammen als Modellierungsartefakt für Systemzustände innerhalb von Modellen,
- D. Eingeschränkte Anwendbarkeit und Effektivität aufgrund fehlender Plausibilität der generierten Testdaten.

Zur Adressierung dieser Einschränkungen werden die Prinzipien des evolutionären Strukturtests auf Modelle angewandt. Der evolutionäre Strukturtest ist ein dynamisches, suchbasiertes Testdatengenerierungsverfahren. Es verwendet evolutionäre Algorithmen für die Suche nach relevanten Testdaten zur iterativen Maximierung der strukturellen Überdeckung des Testobjekts. Durch die Verwendung eines solchen robusten, metaheuristischen Suchverfahrens anstelle von symbolischer Ausführung und des Constraint Solvings, entfallen auch die Einschränkungen dieser Verfahren (Einschränkung A). So können auch jene Testziele erreicht werden, welche von der Erfüllung komplexer Prädikate oder von der Einhaltung aufwändiger Constraints abhängen (Einschränkung B).

*Ansatz:
Evolut.
Testen*

Die Zustände eines Systems werden häufig unter Verwendung von Zustandsübergangsdiagrammen modelliert. Die Strukturüberdeckungskriterien können auch die Überdeckung struktureller Elemente dieser Diagramme fordern, wie beispielsweise die Überdeckung von Transitionsbedingungen. Aus diesem Grund muss bei der strukturorientierten Testdatengenerierung für Modelle ebenfalls eine Überdeckung der eingebetteten Zustandsübergangsdiagramme berücksichtigt werden. Die Anwendung der Prinzipien des evolutionären Strukturtests zu diesem Zweck erscheint möglich und soll in dieser Arbeit untersucht und realisiert werden (Einschränkung C).

*Berücksichtigung von
Statecharts*

Diese Arbeit stellt ein suchbasiertes Verfahren vor, welches Signale (in Form von wert- und zeitdiskreten Datensequenzen) erstellt und diese in Richtung der Erfüllung des Testziels (strukturelle Überdeckung) optimiert. Bei dieser Suche werden Anforderungen des Testers an die zu erstellenden Testdaten berücksichtigt. Bei diesen Anforderungen handelt es sich beispielsweise um Amplitudenbeschränkungen oder aber auch um komplexe Signal Constraints zwischen verschiedenen Eingaben. Dadurch ist es möglich, realitätsnahe und plausible, d.h. für das jeweilige Testobjekt und Testziel sinnvolle Testdaten zu bestimmen (Einschränkungen D).

*Realistische
Testdaten*

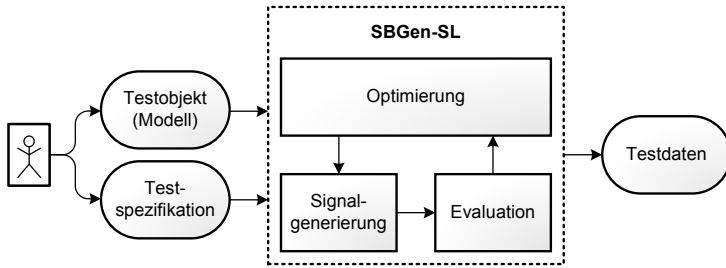


Abb. 1: Übersicht über den in dieser Arbeit entwickelten Ansatz zur Automatisierung der strukturorientierten Testdatengenerierung für Modelle.

Ansatz Der entwickelte Automatisierungsansatz ist abstrakt in Abb. 1 illustriert. Der Tester stellt das Testobjekt (Modell) und die Anforderungen an die Testdaten (Testspezifikation) zur Verfügung, woraufhin die automatisierte Generierung relevanter Testdaten beginnt. Dazu kommen drei Hauptkomponenten zum Einsatz: Die Komponente *Optimierung* umfasst das verwendete metaheuristische Suchverfahren und realisiert die Suche nach relevanten Testdaten. Der *Signalgenerierung* ermöglicht die Erstellung plausibler Signale und *Evaluation* führt das Testobjekt iterativ mit den generierten Signalen aus und misst die Annäherung an die zugrundeliegende Zielstellung. Die folgenden Kapitel gehen auf diese einzelnen Komponenten im Detail ein.

Auswahl von Repräsentanten Als Repräsentant für modellbasierte, grafische Programmiersprachen wurde die ML/SL/SF Entwicklungsumgebung aufgrund ihrer Relevanz im industriellen Umfeld ausgewählt. Die Ideen des vorgestellten Ansatzes lassen sich ebenfalls auf ähnliche Modellierungsverfahren anwenden.

1.2 BEITRÄGE

Die Beiträge dieser Arbeit sind:

1. Die Untersuchung der Besonderheiten des suchbasierten Modell-Strukturtests;

2. Die Analyse des aktuellen Stands der Technik automatischer Testdatengenerierungsverfahren für Modelle und eine Aufarbeitung ihrer Einschränkungen;
3. Der Vorschlag eines suchbasierten Testdatengenerierungsansatzes zur Erfüllung strukturorientierter Überdeckungskriterien für Modelle, bestehend aus
 - a) einer Methode zur Generierung realitätsnaher und plausibler Eingabesignale für das zu testende Modell,
 - b) der Anwendung der Prinzipien der suchbasierten Testdatengenerierung auf SL Modelle und
 - c) der Anwendung der Prinzipien der suchbasierten Testdatengenerierung auf SF Diagramme.
4. Die Demonstration der Effektivität des vorgeschlagenen Ansatzes mit Hilfe einer experimentellen Fallstudie.

1.3 STRUKTUR

Diese Dissertation ist wie folgt aufgebaut:

Kapitel 2 – Grundlagen und verwandte Arbeiten – definiert den terminologischen und methodischen Rahmen dieser Arbeit. Zuerst wird der Modell-Strukturtest eingeführt, mitsamt einer Einführung von Simulationsmodellen, einem Überblick über die Eingliederung des Modelltests in das Gebiet des Softwaretests und einer Spezifikation strukturorientierter Testverfahren. Darauf aufbauend werden verschiedene Verfahren zur Automatisierung des Modell-Strukturtests diskutiert. Anschließend wird genauer auf evolutionäre Algorithmen als Suchverfahren für den Modell-Strukturtest eingegangen.

Kapitel 3 – Signalgenerierung und -optimierung – beschreibt die für den suchbasierten Test von SL/SF Modellen notwendige Generierung und Optimierung von Testdaten in Form von kontinuierlichen Signalen. Es werden drei unterschiedliche Verfahren diskutiert und deren Eignung für den Modell-Strukturtest mit Hilfe eines empirischen Vergleichs untersucht. Darüber hinaus wird ein Verfahren zur Berücksichtigung von Nebenbedingungen für

die zu generierenden Signale (Signal Constraints) vorgestellt. Diese werden unter Verwendung einer temporalen Logik spezifiziert und in den Suchprozess integriert.

Kapitel 4 – Überdeckung von Simulink/Stateflow – gibt einen Überblick über die im Rahmen des Modell-Strukturtests von SL/SF Modellen notwendige Zielfunktionsberechnung. Die definierte Zielfunktion dient der Bewertung der Testdaten bzgl. ihrer Eigenschaft, das Testziel zu erreichen und basiert einerseits auf Beobachtungen des Modellverhaltens während der Simulation und andererseits auf unterschiedlichen Distanzmetriken, welche für SL und SF gesondert aufgezeigt werden.

Kapitel 5 – Fallstudie – beschreibt die zur Demonstration der Effektivität des entwickelten Ansatzes durchgeführte Fallstudie. Es werden vier Testobjekte verwendet, um den Ansatz empirisch sowohl mit der Zufallssuche als auch mit einem kommerziellen Werkzeug zur Testdatengenerierung zu vergleichen. Bei einem der Testobjekte handelt es sich um ein komplexes Modell aus der Praxis. Als Vergleichskriterium wird unter anderem die jeweils erreichte Bedingungsüberdeckung der Testobjekte herangezogen.

Kapitel 6 – Zusammenfassung und Ausblick – fasst die Ergebnisse dieser Arbeit zusammen, zeigt verbliebene Einschränkungen und Grenzen auf und gibt Hinweise auf weiterführende Arbeiten.

2 | GRUNDLAGEN UND VERWANDTE ARBEITEN

*„Schreib doch mal den
State of the Art auf.“*

— Prof. Dr. Stefan Jähnichen

Dieses Kapitel definiert den terminologischen und methodischen Rahmen dieser Arbeit. Abschnitt 2.1 beschreibt den Modell-Strukturtest: Neben den Besonderheiten von Simulationsmodellen, im Speziellen die für diese Arbeit relevanten ML SL Modelle mit eingebetteten SF Zustandsdiagrammen, wird zudem auf den allgemeinen Testprozess und den notwendigen Modell- und Strukturtest eingegangen. Abschnitt 2.2 gibt einen Überblick über vorhandene Ansätze zur Automatisierung der Testdatengenerierung für derartige Modelle und geht auf deren Einschränkungen ein. Abschnitt 2.3 behandelt die für diese Arbeit relevanten suchbasierten Optimierungsverfahren und Abschnitt 2.4 fasst die Ergebnisse zusammen.

2.1 MODELL-STRUKTURTEST

Der Modell-Strukturtest ist wichtiger Bestandteil einer effektiven Teststrategie für den Test von Modellen [31]. Dieses Unterkapitel beschreibt den Modell-Strukturtest im Detail. Dazu wird in Abschnitt 2.1.1 zunächst auf den Begriff des Modells bzw. des Simulationsmodells eingegangen. Anschließend wird in Abschnitt 2.1.2 die Entwicklungsumgebung MATLAB eingeführt. Nach einer Beschreibung allgemeiner Testprozesse in Abschnitt 2.1.3 adressiert Abschnitt 2.1.4 den Test von Modellen. Hier wird auch genauer auf die Rolle des Modell-Strukturtests innerhalb einer effektiven Teststrategie für den Modelltest eingegangen. Abschließend werden in Abschnitt 2.1.5 die Grundlagen strukturorientierter Testdatengenerierungsverfahren beleuchtet.

2.1.1 SIMULATIONSMODELLE

Dieser Abschnitt geht auf den Begriff des Simulationsmodells ein. Für die Beschreibung eines Simulationsmodells ist ein Verständnis für die Begriffe System, Experiment, Modell und Simulation unerlässlich, welche daher zunächst eingeführt werden.

System Ein *System* ist ein reales Objekt mit klar definierter Systemgrenze. Die Systemgrenze bestimmt eindeutig, was zum System und was zur Umwelt des Systems gehört. Im Automobilbereich stellt beispielsweise ein Fahrzeug ein System dar. Jedes System kann aus weiteren eigenständigen Systemen mit separater Systemgrenze bestehen. Im Falle des Systems „Fahrzeug“ stellt beispielsweise der Motor oder jedes Steuergerät des Fahrzeugs ein System – ein Untersystem – dar. Die Interaktion über Systemgrenzen hinweg findet über Ein- und Ausgabeparameter statt. Ein System ist somit steuer- und beobachtbar [24]: Die Manipulation der Eingabeparameter hat Einfluss auf das Systemverhalten, welches über die Ausgabeparameter beobachtbar ist.

Experiment Diese Variation der beeinflussbaren Eingabeparameter eines Systems zum Schließen auf das Systemverhalten wird als *Experiment* bezeichnet. In Abhängigkeit des Systems und des zu beobachtenden Verhaltens, ist dies jedoch aus Kosten-, Sicherheits- oder moralischen Gründen nicht immer möglich. Darüber hinaus sind auch Experimente für Produktneuentwicklungen nicht möglich, da das zu untersuchende System noch nicht existiert. Zur Umgehung dieser Probleme kommen Modelle zum Einsatz.

Modell Ein *Modell* ist eine vereinfachte Darstellung eines Systems. Es bildet bestimmte Aspekte der Realität vereinfacht ab. Frei nach Minsky beschreibt ein Modell eines Systems bezüglich eines Experiments alles, auf das dieses Experiment angewandt werden kann, um Fragen über das System zu beantworten [98]. Ein Modell kann ein reales Objekt (*physikalische Modell*), eine textuelle Beschreibung eines Systems (*verbales Modell*) oder ein Verständnis über die Funktionsweise eines Systems sein (*mentales Modell*). Ein *mathematisches Modell* hingegen beschreibt die Funktionalität eines Systems mit Hilfe mathematischer Gleichungen. Diese Arbeit beschränkt sich auf mathematische Modelle.

Simulation Eine *Simulation* ist ein Experiment, das an einem Modell durchgeführt wird. Eine Simulation wird immer dann durchgeführt, wenn das Experiment für das untersuchte System nicht möglich

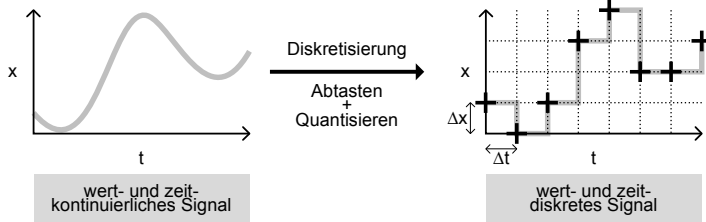


Abb. 2: Diskretisierung eines Signals.

oder der Aufwand dafür zu hoch ist. Ein speziell zum Zweck der Simulation konstruiertes Modell eines Systems wird folgerichtig als *Simulationsmodell* bezeichnet. In dieser Arbeit wird der Begriff „Modell“ synonym für mathematische Simulationsmodelle verwendet.

Eine Simulation wird stets für einen bestimmten Zeitraum durchgeführt. Für jeden Ein- bzw. Ausgabeparameter (sowie interne Größen) existiert für jeden Zeitpunkt innerhalb des Simulationszeitraums ein spezifischer Messwert. Bei Betrachtung mehrerer Zeitpunkte spricht man jeweils von einem *Signal*. Bei fast allen Signalen in der Natur handelt es sich um kontinuierliche Signale, d.h. für jeden beliebigen Zeitpunkt des Simulationszeitraums liegt ein beliebiger Wert der Messgröße vor. Bei der computergestützten Simulation müssen die Signale diskretisiert, d.h. in wert- und zeit-diskrete Signale umgewandelt werden. Dadurch liegen nur noch abzählbar viele Werte als Simulationszeitpunkte und Messgrößen vor. Abb. 2 illustriert diese *Diskretisierung* beispielhaft. Die zeitliche Diskretisierung (Abtasten) wird durch die *Abtastrate* Δt (oder auch Signalauflösung oder Simulationsschrittweite) bestimmt, die Diskretisierung der Datenwerte (Quantisieren) hingegen durch die Granularität des verwendeten Datentyps Δx .

Kontinuierlich vs. diskret

2.1.2 MATLAB/SIMULINK/STATEFLOW

Für diese Arbeit wurde die ML/SL/SF Entwicklungsumgebung aufgrund ihrer Relevanz im industriellen Umfeld als Repräsentant für Modellierungs- und Simulationsumgebungen ausgewählt. Im Folgenden wird daher auf ML, SL und SF eingegangen.

MATLAB

Bei der MATLAB (*Matrix Laboratory*) Entwicklungsumgebung handelt es sich um ein kommerzielles und plattformunabhängiges Werkzeug der Firma The MathworksTM für die numerische Berechnung technischer Problemstellungen, zur Datenanalyse und -visualisierung [126]. Mit der *m language* bietet ML darüber hinaus eine eigene hochentwickelte Programmiersprache zur Definition, Berechnung und Automatisierung mathematischer Funktionsabläufe.

SIMULINK

SL ist eine Erweiterung für ML, welche die modellbasierte Entwicklung dynamischer Systeme ermöglicht und dabei besonders auf die Entwicklung eingebetteter Systeme zugeschnitten ist [34, 129]. Es ermöglicht die Erstellung und Simulation von Modellen. Konzeptuell handelt es sich bei SL Modellen um hierarchische Blockschaltbilder in Form von gerichteten Graphen, bestehen aus mehreren funktionalen Blöcken, welche über ihre Ein- und Ausgabekanäle verbunden sind. Diese Verbindungen definieren den Datenfluss zwischen den Blöcken und beschreiben die mathematischen Gleichungen, die die verwendeten Schnittstellenvariablen zueinander in Relation setzen.

Solver

Die Simulation von SL Modellen wird durch die Verwendung verschiedener kontinuierlicher oder diskreter Gleichungslöser (engl. Solver) realisiert. Diese Solver lassen sich grob in zwei Kategorien einordnen: jene mit fester Abtaste (engl. fixed-step) und jene mit variabler (engl. variable-step) Abtaste. Solver mit fester Abtaste lösen das Modell über den gesamten Simulationszeitraum in konstanten Zeitabständen. Eine kleinere Abtaste führt einerseits durch die höhere Diskretisierung zu einer Erhöhung der Genauigkeit der Ergebnisse, erhöht andererseits aber auch die Simulationszeit. Solver mit variabler Abtaste können demgegenüber die Schrittweite für jeden Zeitschritt in Abhängigkeit von der Dynamik des Modells variieren. Dadurch erhöht sich einerseits der Berechnungsaufwand für jeden einzelnen Zeitschritt, die gesamte Simulationszeit kann andererseits jedoch durch teilweise höhere Schrittweiten verringert werden. Für die Entwicklung eingebetteter Systeme im Automobilbereich kommen Solver mit fester Abtaste zum Einsatz, da dies eine

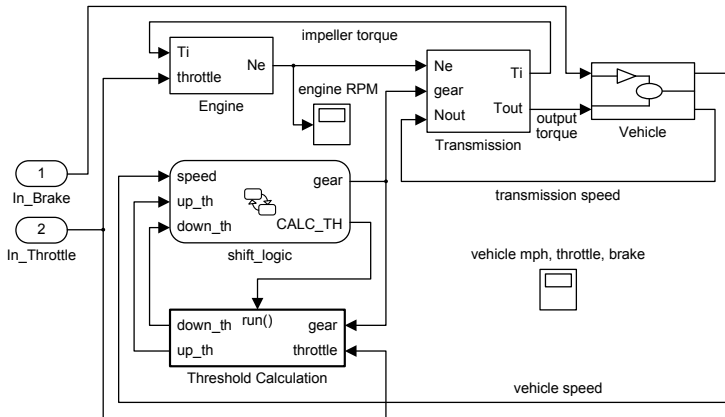


Abb. 3: Simulink Modell einer automatischen Getriebesteuerung.

notwendige Voraussetzung für eine effiziente Codegenerierung darstellt [29].

Durch Verwendung eines Codegenerators kann aus einem SL Modell direkt Programmcode für das jeweilige Zielsystem bzw. die Zielplattform generiert werden. Bei eingebetteter Software handelt es sich überwiegend um C-Code. Die mit SL modellierten Artefakte stehen dadurch in direkter Beziehung zum entwickelten Produkt.

*Automat.
Codegene-
rierung*

Das SL Modell einer automatischen Getriebesteuerung ist beispielhaft in Abb. 3 dargestellt. Zu sehen ist das Modell auf unterster Ebene mit zwei Modelleingaben: *In_Brake* und *In_Throttle*. Die vier rechteckigen Blöcke stellen Untermodelle dar, in denen die jeweilige Funktionalität modelliert ist und welche ihrerseits über diverse Ein- und Ausgaben miteinander verbunden sind. Bei dem Block *shift_logic* handelt es sich um ein SF Diagramm.

Beispiel

Für weitere Informationen, insbesondere bezüglich der Semantik der unterschiedlichen SL Blöcke, sei der Leser auf die SL Dokumentation verwiesen [130].

STATEFLOW

SF ist eine Erweiterung für SL, welche die Entwicklung und die Simulation von Zustandsautomaten und Kontrolllogiken ermöglicht [128]. SF Diagramme sind als einzelne Blöcke in SL Modelle

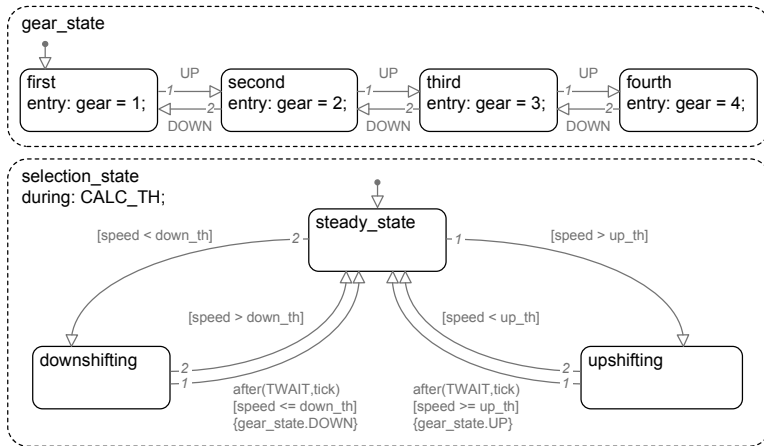


Abb. 4: Stateflow Diagramm einer automatischen Getriebesteuerung.

eingebettet und erlauben dadurch die Verwaltung und die Bestimmung von Systemzuständen. Konzeptuell handelt es sich bei SF Diagrammen um erweiterte endliche Zustandsautomaten basierend auf der von Harel vorgestellten Statechart-Notation [57].

Beispiel

Abb. 4 illustriert beispielhaft das SF Diagramm *shift_logic* des zuvor vorgestellten Modells der automatischen Getriebesteuerung. Dieses ist in zwei parallel ausgeführte Zustände unterteilt: *gear_state* bestimmt den eingelegten Gang des Getriebes und *selection_state* realisiert die Auslösung eines Gangwechsels. Zustandswechsel werden in Abhängigkeit zweier Voraussetzungen durchgeführt: Das Eintreten eines Ereignisses und die Erfüllung der assoziierten Transitionsbedingung (in eckigen Klammern). Eine Transition kann Ereignisse auslösen bei Erfüllung der Transitionsbedingung (in geschweiften Klammern) oder bei tatsächlicher Ausführung der Transition (Angaben nach einem Schrägstrich).

Weitere Informationen, insbesondere bezüglich der Semantik von SF Diagrammen, sind in der SF Dokumentation detailliert beschrieben [135].

2.1.3 SOFTWARETEST ALLGEMEIN

Relevanz

Der *Softwaretest* ist eine wichtige analytische Maßnahme zur Qua-

litätssicherung im Softwareentwicklungsprozess [144]. Testphasen sind wesentliche Bestandteile aller in der Praxis etablierten Vorgehensmodelle zur Softwareentwicklung, wie beispielsweise im Spiralmodell [18] oder im V-Modell [37]. Die systematische Anwendung und Integration von Testmethoden in den Entwicklungsprozess von Software wird – vor allem in sicherheitsrelevanten Anwendungsdomänen – von Sicherheitsrichtlinien und -standards gefordert, z.B. 00-55 Norm [139] in der Rüstungsindustrie, DO-178B Standard [110] in der Luftfahrt und ISO Norm 26262 [71, 72] in der Automobilindustrie.

Vorrangiges Ziel des Testens ist das Entdecken und Lokalisieren von im Testobjekt (das zu testende System) vorhandenen Fehler [137]. Um die Korrektheit der Software zu beweisen, müsste ein erschöpfender Test durchgeführt, d.h. das Testobjekt müsste mit allen möglichen Eingabewerten und all ihren möglichen Kombinationen ausgeführt werden. Da dies jedoch aufgrund der typischerweise enormen Wertebereichsgröße der Eingaben und des daraus resultierenden exponentiellen Aufwands in der Praxis nicht durchführbar ist, kann Testen lediglich das Vorhandensein, und nicht die Abwesenheit von Fehlern aufzeigen [36]. Der Softwaretest ist dennoch eine der wichtigsten Maßnahmen zur Erhöhung des Vertrauens in die Korrektheit und Zuverlässigkeit von Software [35].

Ziel des Testens

Anstelle des erschöpfenden Tests muss ein *Stichprobentest* durchgeführt werden, bei dem das Testobjekt mit einem Satz besonders relevanter Eingabedaten ausgeführt wird. Die Auswahl dieser Eingabedaten, in diesem Zusammenhang als *Testdaten* bezeichnet, gilt als die kritischste Aktivität beim Softwaretest, da sie die Qualität des gesamten Tests maßgeblich beeinflusst. Die Automatisierung der Testdatenauswahl wird als *Testdatengenerierung* bezeichnet.

Testdaten-auswahl

Zur Steuerung der Testdatengenerierung existiert eine Vielzahl von Ansätzen. Man unterscheidet grundlegend zwischen zwei Ausprägungen: Beim *funktionsorientierten Testen* (auch als Black-Box-Test bezeichnet) dient die Spezifikation des Testobjekts als Grundlage für die Auswahl von Testdaten, während die Auswahl der Testdaten beim *strukturorientierten Testen* (auch als White-Box-Test bezeichnet) auf der Implementation des Testobjekts basiert. Ziel des strukturorientierten Testens ist die möglichst vollständige Überdeckung der internen Struktur des Testobjekts. Als *Überdeckung* wird das Verhältnis der während des Testlaufs zur Ausführung gebrachten strukturellen Elemente zur Gesamtheit aller

Black-Box & White-Box

Strukturelemente bezeichnet. Ein Strukturelement wird in diesem Zusammenhang als *überdeckt* bezeichnet, sobald es durch ein Testdatum zur Ausführung gebracht wird. Die Überdeckung dient somit als Indikator für die Eignung und Vollständigkeit eines Testdatensatzes.

2.1.4 MODELLTEST

Ein Test von Modellen wird als *Modelltest* bezeichnet. Der Modelltest kann unterschiedliche *Testziele* verfolgen: Es können grundsätzlich sowohl funktionale als auch nicht funktionale Aspekte getestet werden. So lassen sich beispielsweise funktionale Tests im Rahmen eines Black-Box-Testverfahrens oder auch Strukturtests im Rahmen eines White-Box-Testverfahrens bereits für das Modell durchführen.

*Effektive
Teststrategie*

Eine Kombination beider Verfahren ist im Rahmen einer effektiven Teststrategie – wie aus dem Bereich des klassischen Softwaretests bekannt – besonders zielführend. Nach Grimm [54] sind zunächst spezifikationsbasierte funktionale Tests zur Überprüfung der korrekten Implementation der Anforderungen durchzuführen und die dadurch erreichte strukturelle Überdeckung des Testobjekts zu bestimmen. Zur Überdeckung der noch ausstehenden Strukturelemente sind anschließend gezielte White-Box-Tests (Strukturtests, siehe Abschnitt 2.1.5 auf der nächsten Seite) zu definieren und durchzuführen. Die strukturelle Überdeckung kann bereits parallel während der Durchführung der funktionalen Tests gemessen werden und gibt direkt Aufschluss über die noch nicht getesteten Teile des Testobjekts. Der anschließend durchzuführende Strukturtest ist bei manueller Durchführung besonders bei komplexen Systemen mit sehr hohen Aufwänden verbunden. Eine Automatisierung des Strukturtests ist daher von besonderer Bedeutung. Conrad und Fey haben diese Teststrategie für den Modelltest erweitert [31]. Es kann von der frühzeitigen Simulierbarkeit der Modelle (der Testobjekte) in besonderer Weise profitiert werden. Die im Rahmen einer solchen effektiven Teststrategie durch Kombination funktionsorientierter und strukturorientierter Testverfahren abgeleiteten Testfälle können so anschließend direkt für Back-to-Back-Tests verwendet werden. Der *Back-to-Back-Test* vergleicht das Verhalten des Modells mit dem Verhalten des aus dem Modell generierten Codes bei Ausführung mit dem generierten

Testdatensatz. So kann zusätzlich eine Aussage über die fehlerfreie Konvertierung des Modells in Code und dessen Einbettung in das jeweilige Steuergerät getroffen werden. Die zu erreichende Modellüberdeckung kann darüber hinaus als Stellglied für die gewünschte Testtiefe und als notwendiges Testendekriterium für den funktionalen Modelltest verwendet werden [12]. Baresel *et al.* haben einen direkten Zusammenhang zwischen der erreichten Überdeckung für ein Modell und der Überdeckung für den daraus generierten Programmcode aufgezeigt, sodass die mit Hilfe des Modell-Strukturtest erstellten Testdaten direkt als Ausgangspunkt für folgende Tests des später generierten Programmcodes und dessen Einbettung in das jeweilige Steuergerät verwendet werden können [12].

Da die im Rahmen des Modell-Strukturtests generierten Testdaten als Anreicherung des durch den zuvor durchgeführten funktionalen Test erstellten Testdatensatzes dient, muss insbesondere auf die Generierung plausibler Testdaten geachtet werden. Dazu zählt zum einen das simple Einhalten etwaiger Schranken, wie z.B. Minimal- oder Maximalwerte, zum anderen zählen dazu aber auch mögliche komplexere potentiell wechselseitige Bedingungen zwischen verschiedenen Eingaben. Bei den existierenden Automatisierungsverfahren für den Strukturtest – insbesondere für zustandsbasierte Systeme – fehlt die Beachtung der Plausibilität der generierten Testdaten.

*Plausible
Testdaten*

Aufgrund der potentiellen Zustandsbehaftung von Modellen, müssen die Testdaten mehrere Zeitschritte umfassen. Beim Modell-Strukturtest bestehen die zu generierenden Testdaten daher je Eingabe aus einem Signal einer bestimmten Länge und Signalauflösung – es handelt sich jeweils um wert- und zeitdiskrete Signale.

*Besonderheiten der
Testdaten*

2.1.5 STRUKTURORIENTIERTES TESTEN

Beim strukturorientierten Testen werden relevante Testdaten von der Implementation des Testobjekts abgeleitet. Ziel ist die Erstellung eines Testdatensatzes mit möglichst hohem Überdeckungsgrad des Testobjekts. Eine Ausprägung strukturontierter Testverfahren sind kontrollflussorientierte Verfahren. Diese nehmen Bezug auf den *Kontrollflussgraphen* (engl. Control Flow Graph

Grundlagen des Strukturtests

(CFG)) [61] des Testobjekts, einer grafischen Repräsentation der enthaltenen Kontrollstruktur.

Terminologie der Strukturelemente

Jede Anweisung oder Bedingung des Testobjekts korrespondiert mit einem Knoten des CFG. Der *Kontrollfluss* zwischen zwei Knoten wird im CFG mit Hilfe einer gerichteten Kante zwischen beiden Knoten repräsentiert und als *Zweig* bezeichnet. Ein *Pfad* innerhalb des Graphen ist eine Sequenz aus Knoten und Kanten vom Startknoten zum Endknoten. Verzweigende Knoten, d.h. Knoten von denen mindestens zwei Zweige ausgehen, bilden bedingte Anweisungen ab, Bezug nehmend auf ein *Prädikat*, welches aus mehreren *atomaren Bedingungen* bestehen kann. Die Auswertung eines Prädikats wird als *Entscheidung* bezeichnet. Demgegenüber bilden einfache Knoten im CFG elementare Blöcke ab, d.h. eine Reihe sequentieller Anweisungen. Abb. 5 zeigt den Quellcode eines Beispielprogramms und den korrespondierenden CFG.

Überdeckungskriterien

Welche dieser Strukturelemente in welcher Weise überdeckt werden sollen, wird vom verwendeten *Überdeckungskriterium* des Testverfahrens bestimmt. Die gebräuchlichsten Kriterien sind die Anweisungs-, die Zweig- und die Bedingungsüberdeckung. Jede Anweisung des Testobjekts kann potentiell einen Fehler enthalten, der bei Ausführung dieser Anweisung durch das hervorgerufene Fehlverhalten entdeckt werden kann. Die *Anweisungsüberdeckung* (engl. Statement Coverage (SC)) fordert daher die Ausführung aller Anweisungen des Testobjekts. Die *Zweigüberdeckung* (engl. Branch Coverage (BC)) fordert das Durchlaufen aller Zweige des CFG des Testobjekts, da jeder dieser Zweige potentiell einen Fehler enthalten kann. Dieses Überdeckungskriterium wird auch als *Entscheidungsüberdeckung* (engl. Decision Coverage (DC)) bezeichnet. Die *Bedingungsüberdeckung* (engl. Condition Coverage (CC)) hingegen basiert auf der Annahme des möglichen Vorhandenseins eines Fehlers in jedem Prädikat des Testobjekts. Dieser Fehler wird bei Auswertung der Prädikate mit unterschiedlichen Belegungen ihrer atomaren Bedingungen durch das hervorgerufene Fehlverhalten entdeckt. Es existieren unterschiedliche Varianten der Bedingungsüberdeckung, welche sich in der Art der Kombination der atomaren Bedingungen der Prädikate unterscheiden. Bei der klassischen Bedingungsüberdeckung muss jede atomare Bedingung mindestens ein Mal zu jeweils wahr und falsch ausgewertet werden. Eine weitere häufig eingesetzte Variante ist die modifizierte Bedingungs-/Entscheidungsüberdeckung (engl. Modified Condition/Decision Coverage (MC/DC)) [26]. Diese fordert

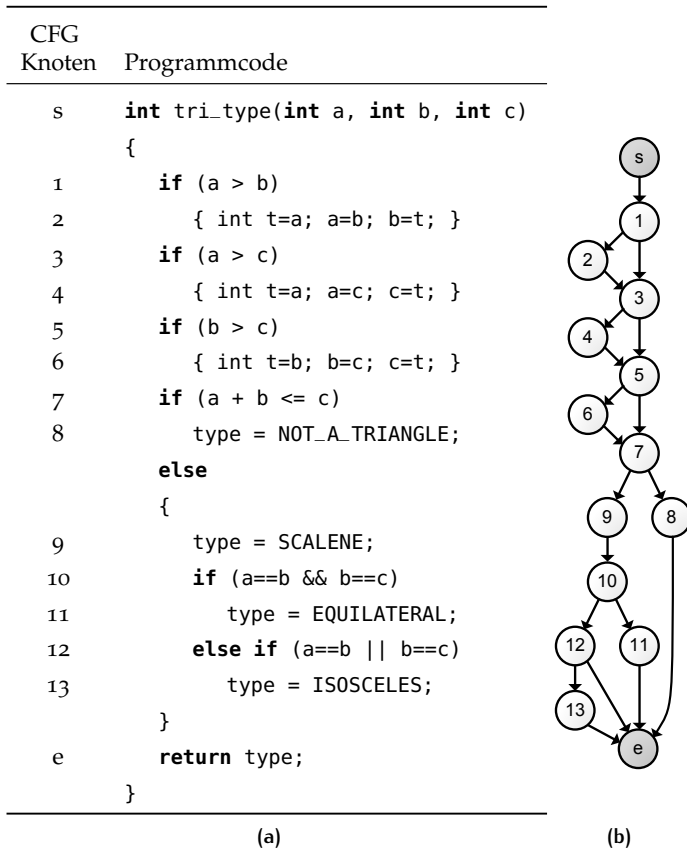


Abb. 5: (a) Ein Beispielprogramm zur Klassifikation von Dreiecken (b) mitsamt des korrespondierenden Kontrollflussgraphen (nach Mc-Minn [93]).

für alle atomaren Bedingungen einer Entscheidung den Nachweis, dass diese einzeln und unabhängig das Ergebnis der Entscheidung beeinflussen. Die unterschiedlichen Überdeckungskriterien wurden von Beizer ausgiebig beschrieben [15].

Ein zu überdeckendes Strukturelement (je nach Überdeckungskriterium eine Anweisung, ein Zweig oder eine Bedingung) wird als *Überdeckungsziel* bezeichnet. Der Einfachheit halber werden in dieser Arbeit einige Redewendungen verwendet, obwohl sie sprachlich nicht korrekt sind: Nähert sich beispielsweise ein Testfall einem Überdeckungsziel an, bedeutet dies, dass sich der Kon-

*Spezielle
Redewen-
dungen*

trollfluss, der aus der Ausführung des Testobjekts mit diesem Testfall resultiert, im Vergleich zu einem vorherigen Zeitschritt näher an der auszuführenden Kontrollstruktur, d.h. der Anweisung, dem Knoten oder dem Zweig befindet. Dies gilt analog für die Distanz eines Testfalls zu einem Überdeckungsziel. Ebenso können Knoten nicht ausgeführt werden, dennoch steht die Ausführung eines Knotens synonym für die Generierung eines Testfalls, der bei Ausführung des Testobjekts zum Durchlaufen des Knotens führt.

*Simulink
Über-
deckung*

Während es sich bei den Überdeckungszielen eines SL Modells um die einzelnen elementaren Rechenblöcke handelt, handelt es sich bei SF Diagrammen um Zustände oder Transitionen bzw. deren Bedingungen. Bei SL Modellen werden alle Blöcke in jedem Zeitschritt der Simulation ausgeführt. Ein Erreichen, d.h. das zur Ausführung bringen, einzelner Blöcke findet somit nicht statt. Dennoch weisen einige SL Blöcke ein verzweigendes Verhalten auf, welches mit dem Verhalten verzweigender Code-Elemente vergleichbar ist. Ein Beispiel ist der *Switch*-Block, bei welchem bereits berechnete Daten nur bei Erfüllung einer bestimmten Bedingung weitergeleitet und anderenfalls verworfen werden. Dies ermöglicht die Anwendung des Zweig- bzw. Entscheidungsüberdeckungstests. Bedingungen und Entscheidungen sind beispielsweise im SL Block *Logical Operator* oder in SF Transitionen gegeben und dienen somit als Grundlage für die Bedingungsüberdeckung. Daher können die zuvor aufgeführten strukturellen Überdeckungskriterien trotz ihrer ursprünglichen Konzeption für den Test prozeduraler Software auch für den Test von SL Modellen angewandt werden [155].

*CC für
SL/SF*

Beispielhaft soll im Folgenden auf die Bedingungsüberdeckung für SL und SF eingegangen werden, da diese als angestrebtes Überdeckungskriterium in den Fallstudien dieser Arbeit verwendet wird (vgl. Kapitel 5 auf Seite 149). Die klassische Bedingungsüberdeckung gibt Aufschluss über das Erreichen aller möglichen Ergebnisse der einzelnen atomaren Bedingungen eines Prädikats: Eine vollständige Bedingungsüberdeckung fordert das Auftreten aller möglichen Ergebnisse dieser atomaren Bedingungen. Besteht ein Prädikat aus n atomaren Bedingungen – die jeweils zu wahr oder falsch ausgewertet werden können – ergeben sich 2^n unterschiedliche Auswertungen, die durch entsprechende Testdaten abzudecken sind. Abb. 6a zeigt beispielhaft für SL einen *Logic*-Block. Zur Überdeckung im Sinne der Bedingungsüberde-

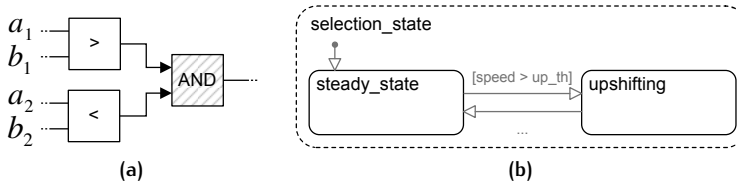


Abb. 6: Beispiele relevanter Strukturelemente für die Bedingungsüberdeckung: (a) Simulink und (b) Stateflow.

ckung müssen die beiden atomaren Bedingungen $a_1 > b_1$ und $a_2 < b_2$ jeweils zu wahr und zu falsch ausgewertet werden, d.h. es muss jeweils ein Testfall zur Erfüllung der vier Bedingungen $a_1 > b_1$, $a_1 \leq b_1$, $a_2 < b_2$ und $a_2 \geq b_2$ gefunden werden. Demgegenüber stellt Abb. 6b die Transitionsbedingung der Transition zwischen den beiden Zuständen `steady_state` und `upshifting` dar (vereinfachte Version des SF Diagramms aus Abb. 4 auf Seite 16). Für deren vollständige Bedingungsüberdeckung muss die Bedingung `speed > up_th` zu wahr und zu falsch ausgewertet werden. Die erstellten Testfälle müssen also die beiden Bedingungen `speed > up_th` und `speed ≤ up_th` überdecken.

2.2 AUTOMATISCHE TESTDATENGENERIERUNG

Eine besonders kritische Aktivität beim Softwaretest ist die Testdatengenerierung (vgl. Abschnitt 2.1.3 auf Seite 16); diese ist besonders aufwändig, fehleranfällig und kostspielig, wenn sie manuell durchgeführt wird. Eine effektive Automatisierung der Testdatengenerierung ist daher besonders wichtig und ist unerlässlich für umfassende Testaktivitäten [121].

Automatisierungsnotwendigkeit

Für die Automatisierung der Testdatengenerierung im Rahmen des strukturorientierten Modelltests wurden diverse Arbeiten vorgestellt. Diese Arbeiten lassen sich grob in *statische* und *dynamische* Verfahren unterteilen. Bei den statischen Verfahren wird das Testobjekt zur Generierung von Testdaten nicht ausgeführt; Testdaten werden stattdessen aus der logischen Struktur berechnet. Dazu kommen die Prinzipien der symbolischen Ausführung und des Constraint Solving zum Einsatz. Diese Prinzipien und ihre An-

statische & dynamische Verfahren

wendung innerhalb statischer Testverfahren für die Testdatengenerierung im Rahmen des strukturorientierten Modelltests werden in Abschnitt 2.2.1 beschrieben. Bei dynamischen Testverfahren wird demgegenüber das Testobjekt zur Generierung von Testdaten zur Ausführung gebracht. Die Aufgabe der Testdatengenerierung wird in ein Suchproblem transformiert, welches unter Verwendung eines geeigneten Suchverfahrens gelöst wird. Abschnitt 2.2.2 geht näher auf diese dynamischen Verfahren ein. Es existiert darüber hinaus eine Reihe von kommerziellen Werkzeugen zur strukturellen Testdatengenerierung, welche in Abschnitt 2.2.3 vorgestellt werden. Die Einschränkungen aller vorgestellten Ansätze werden in Abschnitt 2.2.4 abschließend zusammengefasst.

2.2.1 STATISCHE TESTDATENGENERIERUNG

Im Folgenden wird zunächst auf die Grundprinzipien der statischen Analyse und des Constraint Solvings eingegangen, bevor anschließend die einzelnen Testverfahren beschrieben werden.

SYMBOLISCHE AUSFÜHRUNG UND CONSTRAINT SOLVING

Die symbolische Ausführung zählt zu den statischen Analysemethoden. Erstmals für die Anwendung im Bereich des Softwaretests wurde dieses Verfahren von King und Boyer *et al.* eingesetzt [21, 75]; In diesem Zusammenhang seien ebenfalls Clarke, Howden und Huang erwähnt [27, 67, 68]. In diesen Arbeiten kommt das Prinzip der symbolischen Ausführung zur Anwendung.

Grundprinzip

Das Grundprinzip der symbolischen Ausführung ist, das Testobjekt nicht mit konkreten sondern mit *symbolischen Eingabewerten* auszuführen. Alle Berechnungen innerhalb des Testobjekts werden dabei nicht tatsächlich ausgeführt. Es wird stattdessen unter Verwendung *symbolischer Ausdrücke* ein Prädikatenystem aufgebaut. Das Testobjekt wird Schritt für Schritt emuliert – jeder dieser Schritte beschreibt einen *symbolischen Zustand* des Testobjekts, welcher eine konkrete Programmanweisung repräsentiert und den Pfad zu dieser Programmanweisung in Form des Prädikatensystems enthält. Beim Auftreten einer Verzweigung während der symbolischen Ausführung werden alle (üblicherweise zwei) Zweige durchlaufen. Es entstehen entsprechend zwei neue symbolische

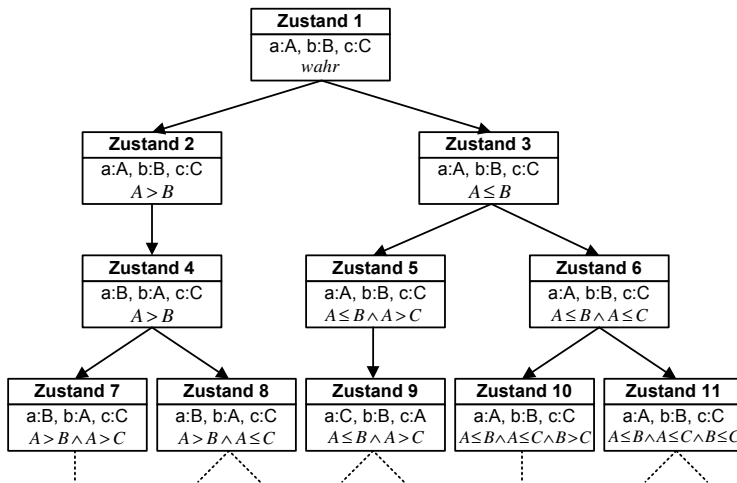


Abb. 7: Die ersten drei Ebenen des durch symbolische Ausführung erstellten Ausführungsbauums des Beispielprogramms aus Abb. 5a auf Seite 21.

Zustände, deren Prädikatsystem mit der jeweiligen Verzweigungsbedingung erweitert wurde. Die symbolische Ausführung erstellt somit für jeden symbolischen Zustand – und somit auch für jedes strukturelle Element des Testobjekts – ein Prädikatsystem, welches einem Satz von Bedingungen für den notwendigen Kontrollfluss zur Erreichung des jeweiligen strukturellen Elements entspricht.

Die symbolische Ausführung eines Programms lässt sich mit Hilfe eines *Ausführungsbauums* visualisieren. Dieser beschreibt den Zusammenhang und die Abhängigkeiten der einzelnen symbolischen Zustände. Die Knoten des Ausführungsbauums repräsentieren jeweils einen symbolischen Zustand, die Kanten zwischen ihnen stellen die möglichen Transitionen dar. Jeder Knoten gibt zum einen die für ihn relevanten symbolischen Eingaben und zum anderen das zur Erreichung des Knoten zu erfüllende Prädikatsystem an. Abb. 7 zeigt die ersten drei Ebenen des Ausführungsbauums für das Beispielprogramm zur Klassifikation von Dreiecken aus Abb. 5a auf Seite 21. Der Wurzelknoten ist der initiale symbolische Zustand (Zustand 1) und entspricht der ersten Anwendung des Programms (CFG-Knoten 1). Den Variablen a , b und c werden die symbolischen Werte A , B und C zugewiesen und das

Ausführungsbauum

Beispiel

initiale Prädikaten-System ist wahr, da an das Erreichen dieses Zustands keine Bedingung geknüpft ist. Bei der ersten Anweisung des Beispielprogramms handelt es sich um eine Verzweigung, weshalb diesem symbolischen Zustand die beiden neuen symbolischen Zustände 2 und 3 entspringen. Entweder wird die Bedingung der Verzweigung zu wahr ausgewertet, so ergibt sich Zustand 2, dessen Prädikaten-System entsprechend durch das Prädikat der Verzweigung ($a > b$) erweitert wird. Wird die Bedingung demgegenüber zu falsch ausgewertet, so ergibt sich Zustand 3. Das Prädikaten-System von Zustand 3 wird in diesem Fall durch das inverse Prädikat der Verzweigung ($\neg(a > b) \equiv a \leq b$) erweitert. Der Übergang von Zustand 2 zu Zustand 4 repräsentiert die Programmanweisung in Zeile 2. Das Prädikaten-System von Zustand 4 hat sich nicht verändert, da hier keinerlei bedingte Verzweigungen durchlaufen wurden. Die symbolischen Werte für a , b und c wurden jedoch angepasst. Der Ausführungsbaum würde analog weiter entwickelt werden. Für das Erreichen von Knoten 11 des Kontrollflussgraphen würde sich beispielsweise das Prädikaten-System $A \leq B \wedge A \leq C \wedge B \leq C \wedge A + B > C \wedge (A = B \wedge B = C)$ ergeben.

*Constraint
Solving*

Mit Hilfe von Constraint Solving Techniken würde dieses Prädikaten-System anschließend aufgelöst und somit die notwendigen Eingabedaten erstellt. Häufig bietet sich – wie auch in diesem Beispiel – eine Vereinfachung des vorhandenen Prädikaten-Systems vor Anwendung der Constraint Solving Techniken an.

Die folgenden Abschnitte beschreiben Ansätze für den statischen Strukturtest von Modellen und Automaten, d.h. dem Ableiten strukturorientierter Testdaten aus dem Testobjekt unter Verwendung von statischen Analyseverfahren. Die zuerst aufgeführten Ansätze behandeln Verfahren, welche die Testobjekte (die zu testenden Modelle bzw. Zustandsübergangsdiagramme) in eine alternative Repräsentation transformieren, auf die anschließend das Prinzip der symbolischen Ausführung bzw. des Model Checkings angewandt wird. Der darauf folgende Abschnitt behandelt spezielle Verfahren, welche die symbolische und die konkrete Ausführung zur Generierung relevanter Testdaten kombinieren.

TRANSFORMATION DER REPRÄSENTATION

Diese Arbeiten transformieren die Testobjekte (Modelle) jeweils in eine alternative Repräsentation, um diese dann symbolisch

auszuführen und die zur Bestimmung von relevanten Eingabedaten notwendigen Prädikatsysteme aufzubauen. Alternativ können auch Model Checking Techniken¹ zum Einsatz kommen. In diesem Zusammenhang ist das Model Checking äquivalent zur symbolischen Ausführung und dem Constraint Solving: Zunächst wird eine Modelleigenschaft definiert, welche die Nicht-Erreichbarkeit des jeweiligen Strukturelements ausdrückt. Der Model Checker wird zur Bestimmung eines Gegenbeispiels verwendet, d.h. es wird versucht, den das zu überdeckende Strukturelement repräsentierenden symbolischen Zustand zu erreichen.

Gadkari *et al.* schlagen eine Methode zur strukturellen Testdatengenerierung für SL/SF Modelle unter Verwendung von Model Checking Techniken vor [44]. In diesem als *AutoMOTGen* [45] bezeichneten Ansatz werden die SL/SF Modelle in ein für den verwendeten Model Checker greifbares formales Modell der Sprache SAL² [120] transformiert. Bei dieser Transformation wird lediglich eine Untermenge der vorhandenen Blöcke und Funktionalitäten von SL und SF unterstützt. Jeder Block wird unter Beachtung der Blockhierarchie und der vorhandenen Datenflussabhängigkeiten in einen einzelnen SAL Block transformiert. Der einzige manuelle Transformationsschritt besteht in der Bereitstellung eines Mappings aller reellwertiger Datenwerte auf ganzzahlige Werte, da der verwendete SAL Model Checker lediglich jene ganzzahligen Werte unterstützt. Das SAL Modell wird unter Beachtung des zu erfüllenden Überdeckungskriteriums des SL/SF Modells mit *Trap*-Variablen instrumentiert. *Trap*-Variablen sind instrumentierte Variablen, welche Rückschlüsse auf die Ausführung bzw. das Erreichen eines bestimmten strukturellen Elements innerhalb des Testobjekts ermöglichen [47]. Der vorgestellte Ansatz ist nicht in der Lage mit nicht-linearen Constraints umzugehen und die Autoren verdeutlichen das Auftreten von Skalierungsproblemen bei Anwendung auf komplexe Testobjekte. Zudem fehlt eine Beschreibung, wie aus dem durch die symbolische Ausführung bestimmten Prädikatsystem konkrete Testdaten – im Falle von SL Signalverläufe – erstellt werden.

Einen ähnlichen Weg gehen Păsăreanu *et al.* mit der Transformation von SL Modellen und UML Zustandsübergangsdiagrammen

-
- 1 Model Checking ist ein statisches Analyseverfahren zur automatischen Identifizierung eventueller Spezifikationsabweichungen eines Modells.
 - 2 SAL (Symbolic Analysis Laboratory) ist ein Framework zur Spezifikation nebenläufiger Systeme, welches dafür einen Satz moderner Model Checker zur Verfügung stellt.

in Java Code [109]. Der aus dem Testobjekt generierte Java Code wird analysiert und somit zur Generierung relevanter Testdaten verwendet. Zu diesem Zweck kommt der Model Checker *Symbolic PathFinder*³ zum Einsatz, welches ebenfalls auf symbolischer Ausführung und Constraint Solving basiert. Dieser Model Checker weist jedoch laut der von den Autoren durchgeführten Experimente Skalierungsprobleme auf und hat ebenfalls Probleme, speziell bei komplexen Testobjekten eine hohe strukturelle Überdeckung zu erreichen [143].

Darüber hinaus transformieren Alur *et al.* SL Modelle und SF Zustandsübergangsdiagramme in hybride Modelle, um diese mit statischen Verfahren zu analysieren [4] (vgl. Agrawal *et al.* [1]). Das Verfahren verfolgt nicht das Ziel der strukturorientierten Testdatengenerierung, sondern vielmehr die Optimierung der Effizienz dynamischer Tests durch Vermeidung redundanter Simulationsläufe. Das in ein hybrides Modell transformierte Testobjekt wird analysiert und für jede Systemeingabe i wird automatisch ein Satz I weiterer äquivalenter Eingaben generiert, welche die selbe Simulation zur Folge haben. Alle so bestimmten Eingaben $i \in I$ können in folgenden dynamischen Tests ausgespart werden. Derartige Einsparungen führen zu einer Effizienzsteigerung des dynamischen Tests.

Mit dem Ziel der symbolischen Verifikation existieren darüber hinaus Verfahren zur Transformation von SL/SF Modellen nach Lustre⁴ [138] oder nach SMV⁵ [8]. Ferner existiert eine Reihe statischer Verfahren für erweiterte endliche Automaten: Chanson und Zhu leiten von den zu testenden Automaten Ausführungspfade ab, führen diese symbolisch aus und erstellen durch Constraint Solving relevante Eingabedaten [25]. Bourhfir *et al.* erweitern diesen Ansatz dahingehend, dass lediglich erreichbare Ausführungspfade im Testdatengenerierungsprozess betrachtet werden [20]. Hong *et al.* transformieren das Testobjekt in ein SMV-Programm, um mit Hilfe eines Model Checkers die durch temporale Logiken ausgedrückten Überdeckungskriterien zu erfüllen [65, 66]. Mit dem Ziel der Testdatengenerierung führt ferner das von Bigot *et al.* vorgestellte Verfahren *AGATHA* eine Transformation der

³ *Symbolic PathFinder* ist eine Erweiterung des Model Checkers *Java PathFinder* [143].

⁴ Lustre ist eine deklarative Programmiersprache, die vor allem zur Entwicklung reaktiver Systeme verwendet wird.

⁵ SMV (Symbolic Model Verifier) ist ein Model Checker mit eigener Syntax und Semantik zur Modellierung von endlichen Automaten.

Testobjekte in eine eigene Formalisierung (AGATHA Extended Input Output Labeled Transition System) durch [17, 87].

CONCOLIC TESTING

Eine weitere Herangehensweise zur Generierung von Testdaten ist die Kombination statischer und dynamischer Analyseverfahren (vgl. Abschnitt 2.2.2 auf Seite 31). Die Stärken der symbolischen (engl. *symbolic*) und der konkreten (engl. *concrete*) Ausführung werden im sogenannten *Concolic Testing* kombiniert. Concolic Testing wird im Rahmen dieser Arbeit als statisches Testdatengenerierungsverfahren aufgeführt, da dabei die symbolische Ausführung und das Constraint Solving im Vordergrund stehen, auch wenn konkrete Ausführungen – wie bei der dynamischen Testdatengenerierung üblich – verwendet werden.

Terminologie

In der Praxis haben statische Testdatengenerierungsverfahren das Problem, dass die durch symbolische Ausführung aufgebauten Prädikatenysteme derart komplex werden, dass sie von den heutigen Constraint Solvern nicht gelöst werden können. Dieses Problem soll beim Concolic Testing abgemildert werden, indem das Testobjekt dynamisch mit konkreten Eingabewerten ausgeführt wird. Diese Eingabewerte werden entweder zufällig erstellt oder vom Benutzer bestimmt. Während der konkreten Ausführung wird das Prädikatenystem für den durchlaufenen Ausführungspfad erstellt. Anschließend wird dieses Prädikatenystem systematisch verändert, z.B. durch Negieren einzelner Teilbedingungen, und mit Hilfe eines Constraint Solvers gelöst. Dadurch können neue konkrete Eingabewerte bestimmt werden, die nun bei Ausführung andere Ausführungspfade des Testobjekts durchlaufen. Durch wiederholte Anwendung dieses Prinzips kann potentiell ein Großteil aller möglichen Pfade zur Ausführung gebracht und die dazu notwendigen Testdaten bestimmt werden.

Motivation

Erstmalig angewandt wurde Concolic Testing von Larson und Austin für die automatische Testdatengenerierung zur Identifikation von Indexüberschreitungen innerhalb von prozeduralem Code [82]. Bei diesem Ansatz wird das Testobjekt mit vom Benutzer bereitgestellten Eingaben ausgeführt. Für den resultierenden Ausführungspfad wird das symbolische Prädikatenystem bestimmt und – falls möglich – von einem Constraint Solver gelöst. Dadurch sollen andere Eingaben für den selben Ausführungspfad

Historie

bestimmt werden, um zu prüfen, ob diese zu einem Fehlverhalten führen könnten. Dieses Vorgehen senkt den Aufwand eines reinen statischen Testdatengenerierungsverfahrens, welches alle möglichen Ausführungspfade betrachten würde.

DART Godefroid *et al.* haben dieses Prinzip in ihrem Verfahren *DART* (Directed Automated Random Testing) zur automatisierten Testdatengenerierung für prozeduralen Code erweitert [50]. Erstmals wurden die während der konkreten Ausführung gesammelten Prädikatsysteme modifiziert und gelöst, um dadurch andere Ausführungspfade zu durchlaufen. Dazu werden systematisch einzelne Teilbedingungen des Prädikatsystems negiert. Das Verfahren verringert die Einschränkungen der verwendeten Constraint Solver, da jene symbolischen Teilbedingungen, die der Constraint Solver nicht lösen kann, durch konkrete Wertebelegungen ersetzt werden können.

CUTE Sen *et al.* stellen mit *CUTE* (A Concolic Unit Testing Engine for C) eine Erweiterung von *DART* vor, welche zusätzlich die Verwendung von Zeigern im Testobjekt unterstützt, so auch Aliasing und Zeiger auf Datenstrukturen [118].

HCT Majumdar und Sen stellen eine Erweiterung von *CUTE* vor: *HCT* (Hybrid Concolic Testing) [88]. Diese führt so lange eine Zufallsuche durch, bis diese stagniert, um anschließend von diesem Punkt aus eine symbolische Ausführung durchzuführen. Die Zufallsuche sorgt in diesem Zusammenhang für eine Breitensuche mit einer anschließenden Tiefensuche durch symbolische Ausführung.

REDIRECT Während *DART*, *CUTE* und *HCT* auf die Unterstützung der Programmiersprachen C und Java ausgelegt sind, werden deren Prinzipien mit dem Verfahren *REDIRECT* (Randomized Directed Testing) von Satpathy *et al.* erstmals auf ML/SL Modelle angewandt [115]. Das Verfahren versucht, die Stärken der Zufallsuche, der symbolischen Ausführung und des Constraint Solvings zu vereinen. Zusätzlich werden einzelne nicht-lineare SL Blöcke durch Anwendung eines einfachen Ansatzes zum heuristischen und musterbasierten Lösen von Constraints unterstützt. Die SL Modelle werden zur Anwendung eines Constraint Solvers wie bei AutoMOTGen in SAL Modelle transformiert. Für SF Diagramme muss die Parallelität und Hierarchie der Zustände aufgelöst werden, was zu einem enormen Anstieg der Transitionen und der Komplexität des Diagramms führt.

Alle vorgenannten Concolic Testing Ansätze haben das Problem, dass die verwendeten Constraint Solver für das Lösen der entstehenden Prädikatsysteme trotz Vereinfachung durch konkrete Ausführungen bei Anwendung auf komplexe Testobjekte zu schwach sind [117]. Die Skalierbarkeit der Verfahren ist nach wie vor das hauptsächliche Hindernis für die Anwendung der symbolischen Ausführung [108].

Grenzen

2.2.2 DYNAMISCHE TESTDATENGENERIERUNG

Im Vergleich zu den vorgestellten statischen Testverfahren sind die dynamischen Testverfahren hauptsächlich durch die Ausführung des Testobjekts mit konkreten Eingabewerten charakterisiert. Während der Ausführung wird das Verhalten des Testobjekts beobachtet und zur Bewertung der vorgeschlagenen Eingaben genutzt. Die Eingabewerte werden somit nicht formal aus der Struktur des Testobjekts berechnet, sondern mit Hilfe eines Suchverfahrens gesucht. Durch Anwendung dieses suchbasierten Tests sollen die Grenzen der symbolischen Ausführung und des notwendigen Constraint Solvings überwunden werden [78]. Im Folgenden wird zunächst das Prinzip des suchbasierten Strukturtests vorgestellt, anschließend wird auf einzelne suchbasierte Ansätze für den strukturorientierten Modelltest eingegangen.

DER SUCHBASIERTE STRUKTURTEST

Einen umfangreichen und vollständigen Überblick über die historische Entwicklung des suchbasierten Strukturtests seit Mitte der 1970er Jahre liefert die Arbeit von McMinn [93]. Die Arbeiten konzentrieren sich hauptsächlich auf den Einsatz suchbasierter Strukturtests für prozeduralen Code. Seit dem Jahr 2000 wird vermehrt die Unterstützung der Programmiersprache C [70] vorangetrieben, aufgrund ihrer Verbreitung im industriellen Umfeld für die Entwicklung eingebetteter Systeme.

Historie

Der suchbasierte Strukturtest transformiert die Aufgabe der strukturorientierten Testdatengenerierung in ein Optimierungsproblem, welches unter Verwendung eines Suchverfahrens gelöst wird. Die Anwendung eines Suchverfahrens im Rahmen des suchbasierten Tests erfordert zum einen die Definition des Suchraums. Der *Suchraum* charakterisiert eine spezifische Menge von Punkten sowie

Prinzip

deren Definitionsbereich. Beim suchbasierten Strukturtest handelt es sich bei diesen Punkten um Testeingaben zur Ausführung des Testobjekts. Als *Repräsentation* wird in diesem Zusammenhang die Definition der Kodierung der konkreten Testeingaben in Datenstrukturen bezeichnet, die vom eingesetzten Suchverfahren verarbeitet werden können. Zum anderen wird die Definition einer Zielfunktion benötigt: Die *Zielfunktion* bewertet die generierten Testeingaben bezüglich ihrer Fähigkeit, ein gegebenes Strukturelement zu überdecken. Dazu ist eine Messung des Daten- bzw. Kontrollflusses innerhalb des Testobjekts während der Testausführung notwendig. Diese Messung erfolgt durch Auslesen interner Wertebelegungen mit Hilfe zuvor eingefügter protokollierender Anweisungen. Das Einfügen dieser das Verhalten des Testobjekts nicht beeinflussender Anweisungen wird als *Instrumentierung* bezeichnet. Näher auf die Terminologie und Konzepte der Repräsentationen und der Zielfunktionen wird in Kapitel 2.3 auf Seite 48 eingegangen.

Suchraum Beim Suchraum unterscheidet man in einen phänotypischen und einen genotypischen Suchraum (vgl. Abschnitt 2.3 auf Seite 48). Der phänotypische Suchraum beschreibt den Raum jener Datenstrukturen, die der Schnittstelle des Testobjekts entsprechen. Für die in Abb. 5 auf Seite 21 dargestellte Funktion ergibt sich aus den drei Eingabevariablen a , b und c der phänotypische Suchraum $D_{\text{int}} \times D_{\text{int}} \times D_{\text{int}}$. D_{int} ist der Definitionsbereich des Datentyps Integer. Der phänotypische Suchraum beschreibt daher die Menge aller möglichen Eingaben für die Funktion `tri_type`. Demgegenüber beschreibt der genotypische Suchraum die Schnittstelle zum angewandten Suchverfahren. Da für den suchbasierten Strukturtest für eine Vielzahl elementarer Datentypen Variationsoperatoren zur Verfügung stehen, entspricht im angegebenen Beispiel der phänotypische genau dem genotypischen Suchraum. Dies wäre nicht der Fall, würden vom Testobjekt strukturierte Datentypen, wie beispielsweise `structs` oder `unions`, verwendet werden. In diesem Fall käme der *Decoder* zum Einsatz, der die vom Suchverfahren verarbeitbare Sequenz primitiver Daten in konkrete, zur Ausführung des Testobjekts geeignete Datenstrukturen umwandelt.

Aufteilung in Teilziele Das Testziel der vollständigen strukturellen Überdeckung wird in Teilziele unterteilt, welche für das jeweilige Testobjekt aus dem gewählten und zu erfüllenden Überdeckungskriterium hervorgehen. Ein solches Teilziel wird in dieser Arbeit als *Überdeckungsziel*

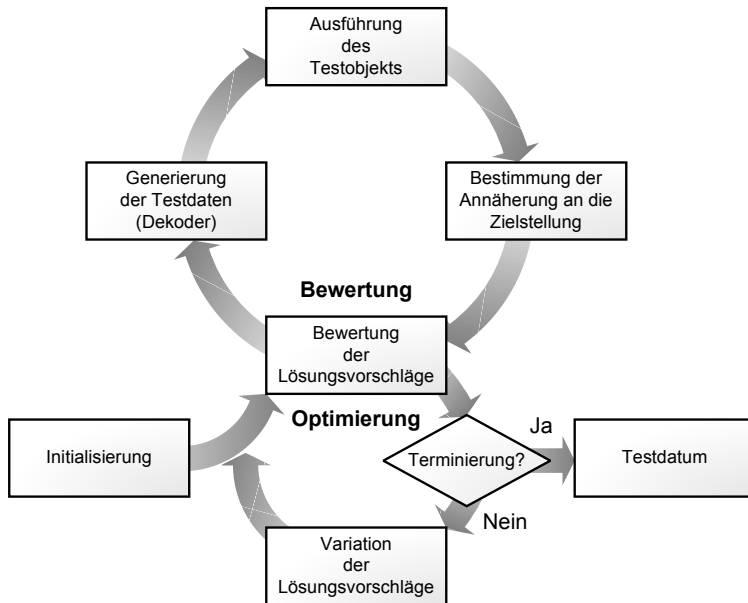


Abb. 8: Allgemeiner Ablauf des Optimierungsprozesses für ein Teilziel im Rahmen des suchbasierten Strukturtests.

bezeichnet. Beispielsweise gilt für die Zweigüberdeckung jeder Zweig des Kontrollflussgraphen des Testobjekts als ein solches Überdeckungsziel, welches es zu erreichen gilt. Die Überdeckungsziele werden separat als Optimierungsproblem aufgefasst und vom verwendeten Optimierungsverfahren gelöst, d.h. es wird für jedes Überdeckungsziel ein gesondertes Testdatum generiert. Im Gesamtprozess der Optimierung wird ein Satz von Testdaten generiert, welcher in seiner Gesamtheit die vollständige Überdeckung des Testobjekts gemäß des gewählten Überdeckungskriteriums annähert. Der Optimierungsprozess eines dieser Überdeckungsziele ist in Abb. 8 dargestellt. Herzstück ist das verwendete Such- bzw. Optimierungsverfahren, welches potentielle Lösungen des Optimierungsproblems erstellt, das Testobjekt damit ausführt und die Bewertungen dieser Lösungen entsprechend berücksichtigt. Die genotypischen Lösungsvorschläge werden zuvor in phänotypische Lösungen (Testdaten) umgewandelt.

Die Bewertung der erzeugten Lösungsvorschläge, d.h. die Bestimmung eines Zielfunktionswertes, erfolgt anhand der (während

*Ziel-
funktion*

der Ausführung gemessenen) Annäherung an das zu erreichende Überdeckungsziel. Für jedes Überdeckungsziel muss derart eine gesonderte Zielfunktion entwickelt werden, sodass das Erreichen ihres globalen Optimums zur Erfüllung dieses Überdeckungsziels führt. Die Zielfunktion liefert für jedes Testdatum einen numerischen Wert, der die Fähigkeit des evaluierten Lösungsvorschlags widerspiegelt, das aktuelle Überdeckungsziel zu erreichen. Die von Wegener *et al.* [146] vorgestellte Zielfunktion gilt für den suchbasierten Strukturtest prozeduralen Codes als der heutige Stand der Technik und besteht aus den im Folgenden erläuterten zwei Distanzmetriken: der *Annäherungsstufe* ϑ und der *Zweigdistanz* δ . Die Zielfunktion $\Phi(t, td)$ für das Überdeckungsziel t und das Testdatum td ist definiert als:

$$\Phi(t, td) = \vartheta + \delta. \quad (2.1)$$

Annäherungsstufe

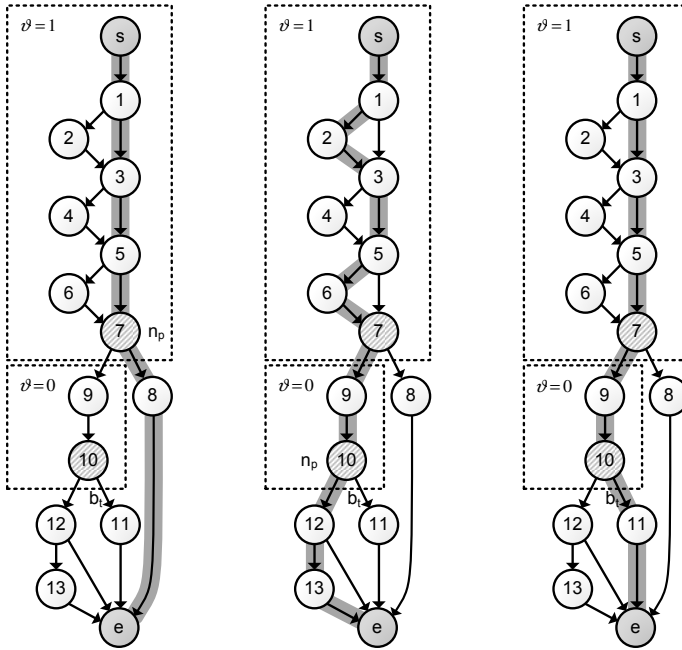
Die Annäherungsstufe $\vartheta \in \mathbb{N} \cup \{0\}$ beschreibt die Distanz des durch die generierten Teststimuli hervorgerufenen Kontrollflusses zur durch das Überdeckungsziel definierten und zu erreichenden Kontrollstruktur. Die Berechnung der Annäherungsstufe basiert auf den so genannten *entscheidenden Verzweigungsknoten*, welche ausgehende Zweige besitzen, deren Ausführung das Erreichen des Zielknotens unmöglich machen. Ein solcher Zweig wird als *entscheidender Zweig* b_d bezüglich des Zielzweigs b_t bezeichnet. Der Quellknoten von b_d liegt auf einem Pfad zu b_t , es existiert jedoch kein Pfad von b_d nach b_t . Ein entscheidender Verzweigungsknoten, dessen entscheidender Zweig durchlaufen wird, wird als *Problemknoten* n_p bezeichnet. Die Annäherungsstufe berechnet sich als die minimale Anzahl entscheidender Verzweigungsknoten auf allen möglichen Pfaden vom Problemknoten n_p zum Zielpfad b_t :

$$\vartheta = \min \left(\left\{ \chi(\pi) \mid \pi \in P_{n_p, b_t} \right\} \right). \quad (2.2)$$

Dabei ist P_{n_p, b_t} die Menge aller Pfade vom Problemknoten n_p zu Zielzweig b_t und $\chi(\pi)$ beschreibt eine Funktion, welche die Anzahl der entscheidenden Verzweigungsknoten innerhalb des Pfades π liefert (Definition nach Wappler [145]).

Beispiel

Abb. 9 illustriert den durch drei unterschiedliche Teststimuli hervorgerufenen Kontrollfluss der Funktion aus Abb. 5 auf Seite 21. Grau hinterlegt ist jeweils der zur Ausführung gebrachte Pfad. Bei den beiden schraffiert dargestellten Knoten (7 und 10) handelt es



(a) Teststimuli: $(\alpha, b, c) = (1, 2, 5)$. (b) Teststimuli: $(\alpha, b, c) = (3, 2, 2)$. (c) Teststimuli: $(\alpha, b, c) = (2, 2, 2)$.

Abb. 9: Drei unterschiedliche Kontrollflussdarstellungen der Funktion aus Abb. 5 auf Seite 21. Umrandet sind die Bereiche der beiden sich ergebenden Annäherungsstufen aufgrund der schraffiert dargestellten entscheidenden Verzweigungsknoten bezüglich Zielzweig b_t .

sich um entscheidende Verzweigungsknoten bezüglich des Zielzweigs b_t zwischen Knoten 10 und 11. Daraus ergeben sich für diesen Zielzweig zwei Annäherungsstufen – hier als umrandete Bereiche dargestellt. Für den in Abbildung 9a dargestellten Kontrollfluss ergibt sich für den Zielzweig b_t die Annäherungsstufe $\vartheta = 1$, dies entspricht genau der minimalen Anzahl entscheidender Verzweigungsknoten zwischen Problemknoten n_p und Zielzweig b_t . Für den Kontrollfluss in Abbildung 9b gilt $\vartheta = 0$, da sich kein entscheidender Verzweigungsknoten zwischen dem Problemknoten und dem Zielzweig befindet. Abbildung 9c zeigt den zum Erreichen des Zielzweigs notwendigen Kontrollfluss; in diesem Fall ergibt sich ebenfalls die Annäherungsstufe $\vartheta = 0$.

*Zweig-
distanz*

Die Annäherungsstufe kann bereits zur Bewertung von Teststimuli bezüglich des Erreichens eines bestimmten Strukturelements genutzt werden. Falls jedoch unterschiedliche Teststimuli zu den selben Problemknoten führen, erhalten diese auch die selbe Annäherungsstufe und sind daher nicht genauer unterscheidbar. Dennoch kann eines der Stimuli in Hinblick auf das vom Problemknoten verwendete Prädikat besser zum Erreichen des Zielzweigs geeignet sein. Zu diesem Zweck wird die Zweigdistanz verwendet. Diese gibt die Distanz zum Erfüllen des Prädikats des Problemknotens an. Die *Zweigdistanz* $\delta(p, b)$ für Prädikat p und gewünschter Prädikatauswertung $b \in \{T, F\}$ ist definiert als:

$$\delta(p, b) = \begin{cases} 0 & \text{falls } E(p) = b \\ \text{norm}(\delta_b(p)) & \text{sonst.} \end{cases} \quad (2.3)$$

Die Funktion E gibt die Prädikatauswertung für das übergebene Prädikat p und den verwendeten Teststimulus zurück und $\delta_b(p)$ ist die relations- bzw. operatorspezifische Distanzfunktion für Prädikat p (Definition nach Wappler [145]).

*Distanz-
funktionen*

Diese relations- bzw. operatorspezifischen Distanzfunktionen berechnen je nach vorliegender Relation oder logischem Operator die Distanz zur gewünschten Prädikatauswertung. Besteht das Prädikat aus mehreren atomaren Bedingungen, so wird die Distanz zu diesen separat berechnet und anschließend zu einer Gesamtdistanz zusammengefasst. Zu jeder Relation existieren damit zwei Distanzfunktionen: eine für den Ergebniswert T und eine für den Wert F : φ_T bzw. φ_F . Die typischerweise eingesetzten Distanzfunktionen für die einzelnen Relationen und logischen Operatoren nach Tracey *et al.* [136] sind in Tabelle 1 auf der nächsten Seite dargestellt. Die berechneten Distanzen werden nach Arcuri mit der Funktion $\text{norm}(d) = d \cdot (d + \beta)^{-1}$ mit $\beta > 0$ auf das Intervall $[0, 1)$ skaliert, um sicherzustellen, dass der größtmögliche Wert kleiner ist, als der kleinstmögliche Wert der Abstandsstufe [11, 94]. Die Distanzfunktionen für logische Operatoren sind der Arbeit von Bottaci entnommen [19]. In der ersten Spalte der Tabelle sind die Bezeichnungen der einzelnen spezifischen Distanzfunktionen aufgeführt. Die zweite Spalte definiert die jeweilige Distanzfunktion, wenn das gegebene Prädikat zu T ausgewertet werden soll, während die Distanzfunktionen für eine Auswertung zu F in der dritten Spalte definiert werden. In den Definitionen ist κ der kleinstmögliche Wert des verwendeten Datentyps (üblicherweise $0 < \kappa \ll 1$).

	T erwünscht (δ_T)	F erwünscht (δ_F)
$\delta_{x < y}$	$x - y + \kappa$	$\delta_{x \geq y}$
$\delta_{x > y}$	$y - x + \kappa$	$\delta_{x \leq y}$
$\delta_{x \leq y}$	$x - y$	$\delta_{x > y}$
$\delta_{x \geq y}$	$y - x$	$\delta_{x < y}$
$\delta_{x == y}$	$ x - y $	$\delta_{x \neq y}$
$\delta_{x \neq y}$	1	$\delta_{x == y}$
$\delta_{e_1 \& e_2}$	$\delta_T(e_1) + \delta_T(e_2)$	$\frac{\delta_F(e_1) \cdot \delta_F(e_2)}{\delta_F(e_1) + \delta_F(e_2)}$
$\delta_{e_1 e_2}$	$\frac{\delta_T(e_1) \cdot \delta_T(e_2)}{\delta_T(e_1) + \delta_T(e_2)}$	$\delta_F(e_1) + \delta_F(e_2)$
δ_{-e}	$\delta_F(e)$	$\delta_T(e)$

Tab. 1: Relations- bzw. operatorspezifische Distanzfunktionen

Zur Lösung des Optimierungsproblems kommen metaheuristische Optimierungsverfahren zum Einsatz, da diese besonders für komplexe, hochdimensionale Suchräume geeignet sind, für die keinerlei Gradienteninformationen zur Verfügung stehen. Üblicherweise kommen evolutionäre Algorithmen (vgl. Abschnitt 2.3 auf Seite 48) zum Einsatz; man spricht dann entsprechend vom *evolutionären Strukturtest*.

Die folgenden Abschnitte beschreiben Ansätze für den suchbasierten Strukturtest von Modellen. Der erste Ansatz betrachtet SL Modelle, während die restlichen Ansätze auf die Überdeckung endlicher Automaten abzielen.

ZHAN

Die Arbeiten von Zhan und Clark beschreiben, wie die vorhandenen suchbasierten und struktuorientierten Testverfahren prozeduralen Codes für die Anwendung auf SL Modelle angepasst und zur Generierung geeigneter Testdaten verwendet werden können [153, 154]. Als Testziel wird die Zweigüberdeckung – wie von Reactis vorgeschlagen [112] – angestrebt: Berücksichtigt werden jene SL Blöcke, die ein konditionales Verhalten aufweisen und jeweils zwei unterschiedliche Auswertemöglichkeiten besitzen. Im Speziellen handelt es sich um die Blöcke *LogicalOperator*, und *RelationalOperator*, deren zwei Auswertungen zu T bzw. F, als T-

bzw. F-Zweig interpretiert werden. Analog dazu wird für *Switch*-Blöcke das Weiterleiten des ersten Eingabeports als T- und das Weiterleiten des dritten Eingabeports als F-Zweig interpretiert.

Suchraum

Die vorgestellten Arbeiten unterscheiden nicht zwischen einem phäno- und einem genotypischen Suchraum. Das verwendete Suchverfahren operiert direkt auf den Variablen der Eingabeschnittstelle des getesteten Modells. Die Messung des Datenflusses innerhalb der SL Modelle erfolgt durch das Einfügen sogenannter *Probe*-Blöcke an die Signalverbindungen der Eingabeports des zu überdeckenden Blocks. Diese instrumentierten Blöcke protokollieren den jeweils anliegenden Wert für jeden einzelnen Zeitschritt. Falls der zu überdeckende Block auf booleschen Eingabewerten basiert, werden diese – falls möglich – so weit rückpropagiert, bis unterscheidbare Aussagen getroffen werden können.

Instrumentierung

Relationale Blöcke

Abb. 10a illustriert dieses Vorgehen für einen beispielhaften relationalen Block ($>$). Beide Eingaben zu diesem Block sind für die Erfüllung der Bedingung von Belang und müssen für die Messung der Annäherung protokolliert werden. Bei den beiden Blöcken $p1$ und $p2$ handelt es sich um jene zuvor erwähnten *Probe*-Blöcke. Durch die Messung dieser Eingabewerte während der Laufzeit lässt sich die Annäherung an die Erfüllung des Prädikats $p1 > p2$ zum Erreichen des T-Zweigs bzw. $\neg(p1 > p2) \Rightarrow p1 \leq p2$ zum Erreichen des F-Zweigs berechnen. Ähnlich verhält sich die Instrumentierung für logische Blöcke, wie in Abb. 10b illustriert. Beide Eingabesignale müssen protokolliert werden. Da diese jedoch in diesem Fall boolesche Werte enthalten, müssen sie innerhalb des Modells rückpropagiert werden. Die Eingabesignale, aus denen sich die booleschen Signale ergeben, müssen statt dessen protokolliert werden. Im Beispiel werden die beiden Eingabesignale für den oberen relationalen Block ($>$) mit Hilfe der *Probe*-Blöcke $p1$ und $p2$ überwacht, während die beiden Eingabesignale für den unteren relationalen Block ($<$) mit Hilfe von $p3$ und $p4$ protokolliert werden. Das sich ergebende Prädikat für den T-Zweig des logischen Blocks lautet $p1 > p2 \wedge p3 < p4$; der F-Zweig ist demgegenüber repräsentiert durch das Prädikat $p1 \leq p2 \vee p3 \geq p4$.

Logische Blöcke

Switch-Blöcke

Für *Switch*-Blöcke ist das zweite Eingabesignal in Kombination mit dem ausgewählten Schwellwert θ und dem gewählten Relationsoperator $\otimes$ für die Zweigüberdeckung relevant. Für den Switch-Block aus dem Modellausschnitt in Abb. 10c lauten die zu erfüllenden Prädikate $p1 \otimes \theta$ und $p1 \neg \otimes \theta$ für den T- bzw. den F-Zweig.

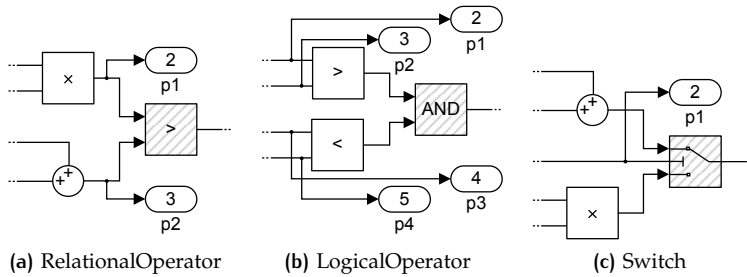


Abb. 10: Angewandte Instrumentierung nach Zhan [153] für unterschiedliche SL Blöcke zur Messung der Annäherung an ein Überdeckungsziel der Zweigüberdeckung.

Das Erreichen eines jeden Überdeckungsziels gilt als gegeben, sobald das Teilziel in mindestens einem Zeitschritt k der Ausführung erfüllt ist. Aus diesem Grund werden die einzelnen Prädikate p_k aller Zeitschritte disjunktiv verknüpft, um dadurch ein Gesamtprädikat für das Erreichen des Überdeckungsziels zu erhalten: $p = p_1 \vee p_2 \vee \dots \vee p_k$. Für dieses Gesamtprädikat wird mit den von Bottaci [19] vorgestellten Distanzfunktionen ein Zielfunktionswert berechnet. Zur Optimierung dieser Zielfunktion kommt das Verfahren *Simulated Annealing* zum Einsatz. Simulated Annealing ist ein metaheuristisches Optimierungsverfahren, welches den Abkühlprozess von glühenden Metallen simuliert, deren Atome durch ein langsames Abkühlen in der Lage sind, optimale Strukturen – und somit einen energetisch günstigen Zustand – einzunehmen [76, 95]. Die mit Hilfe der Zielfunktion bestimmten Bewertungen der Lösungsvorschläge werden grundlegend wie bei einer üblichen lokalen Suche verwendet. Der im Laufe des Optimierungsprozesses linear abgesenkte Parameter *Temperatur* bestimmt die Wahrscheinlichkeit, einen Lösungsvorschlag mit schlechterer Bewertung zur weiteren Bearbeitung zu verwenden. Dies ermöglicht das Verlassen lokaler Optima und erhöht dadurch die Wahrscheinlichkeit das globale Optimum zu finden. Vergleichend dazu verwenden Ghani *et al.* genetische Algorithmen (vgl. Abschnitt 2.3 auf Seite 48) zur Optimierung und weisen deren Überlegenheit gegenüber Simulated Annealing durch empirische Untersuchungen nach [49].

Ziel-
funktion

Opti-
mierung

Das vorgestellte Verfahren dient bislang nur als Nachweis der Konzepttauglichkeit. Es wurde lediglich auf kleine Modelle mit

Grenzen

einer Blockanzahl von 15 bis 69 Blöcken und auf einen sehr limitierten Satz verzweigender SL Blöcke angewendet und die in industriellen Modellen sehr häufig zum Einsatz kommenden SF Diagramme finden ebenfalls keine Berücksichtigung. Ebenso können die für die unterschiedlichen Überdeckungsziele notwendigen Prädikate bei langen Simulationszeiten und der damit einhergehenden hohen Anzahl an Zeitschritten sehr komplex werden.

UNITED

Norbert Oster stellt ein Verfahren zur automatischen Testdatengenerierung für objektorientierte Programmiersprachen vor [101]. Dieses zielt auf die Maximierung der erreichten Überdeckung des Testobjekts bei gleichzeitiger Minimierung der dazu notwendigen Anzahl an Testdaten ab. Dies wird durch den Einsatz eines mehrkriteriellen Optimierungsverfahrens ermöglicht: In die Bewertung der generierten Lösungsvorschläge, welche hier jeweils einem gesamten Testdatensatz entsprechen, fließt einerseits deren erreichte strukturelle Überdeckung und andererseits die Größe des Testdatensatzes ein.

Im Rahmen des Förderprojektes *UnITeD* (Unterstützung inkrementeller Testdaten) wird dieses Prinzip von Florin Pinte *et al.* auf UML-Zustandsübergangsdiagramme angewandt [40, 102]. Für das Erreichen eines bestimmten Testziels, d.h. eines bestimmten strukturellen Elements des Testobjekts, wird bei diesem Verfahren lediglich die Überdeckung des gesamten Testobjekts für die Zielfunktion verwendet. Es findet daher keine zielgerichtete Suche nach Testdaten statt – dies gilt insbesondere für schwer zu erreichende Testziele. Das Prinzip der Pareto-Optimierung verstärkt dieses Problem zusätzlich, weil – selbst bei entsprechenden Priorisierungen – stets ein Kompromiss aus beiden Optimierungskriterien erzielt wird. Eine Maximierung der strukturellen Überdeckung ist somit nicht garantiert.

PFADÜBERDECKUNG FÜR AUTOMATEN

Es existieren Arbeiten, die sich mit der suchbasierten Testdatengenerierung zur Überdeckung von Ausführungspfaden für erweiterte endliche Automaten befassen. Diese sollen hier ebenfalls Erwähnung finden, da deren grundlegenden Prinzipien auch auf andere Überdeckungskriterien angewandt werden können.

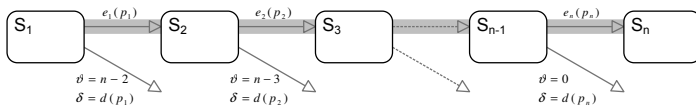


Abb. 11: Bewertung eines Ausführungspfads bzgl. des Durchlaufens des grau hinterlegten Zielpfades innerhalb eines Zustandsübergangsdiagramms (nach Lefticaru und Ipate [83]).

Lefticaru und Ipate stellen ein Verfahren vor, welches durch Anwendung der ursprünglich für Kontrollflussgraphen konzipierten, von Wegener *et al.* vorgeschlagenen Zielfunktion, basierend auf den Distanzmetriken Annäherungsstufe und Zweigdistanz (vgl. Abschnitt 2.2.2 auf Seite 31), Testdaten für einen gegebenen Ausführungspfad ableitet [83, 84]. In einem ersten Schritt wird gemäß des gewünschten Überdeckungskriteriums ein Satz von Ausführungspfaden durch das Zustandsübergangsdiagramm bestimmt. Für all diese Pfade wird im folgenden Schritt versucht, jeweils durch Anwendung genetischer Algorithmen ein Testdatum zu generiert, welches diesen Zielpfad durchläuft.

Die vorgestellte Zielfunktion drückt die Annäherung des Ausführungspfads (korrespondierend zum zu bewertenden Testdatum) an den Zielpfad aus. Die Zielfunktion besteht zum einen aus der Annäherungsstufe ϑ , welche der Anzahl der Zustände zwischen dem Zielzustand und dem Problemzustand (analog zum Problemknoten eines CFG) entspricht. Darüber hinaus wird zum anderen die Zweigdistanz δ zur Berechnung des Zielfunktionswertes verwendet. Diese beschreibt die Distanz zum Auslösen jenes Ereignisses, welches am Problemzustand die Transition auf dem Zielpfad durchlaufen lässt. Abb. 11 illustriert dieses Prinzip. Der die Zustände s_1 bis s_n durchlaufende und grau hinterlegte Zielpfad soll beispielhaft überdeckt werden. Divergiert der aktuell zu bewertende Ausführungspfad beispielsweise nach Zustand s_2 , so befinden sich auf dem Zielpfad $n - 3$ Zustände zwischen diesem Problemzustand und dem Zielzustand s_n ; es gilt $\vartheta = n - 3$. Die Zweigdistanz basiert auf der Transitionsbedingung p_2 und wird mit Hilfe einer Distanzfunktion d berechnet; es gilt: $\delta = d(p_2)$. Die Autoren verwenden für d die von Tracey *et al.* vorgeschlagenen Distanzfunktionen. Divergiert der Ausführungspfad demgegenüber nach Zustand s_{n-1} , so befindet sich auf dem Zielpfad kein Zustand zwischen Problem- und Endzustand; es gilt dann: $\vartheta = 0$. Gilt $\vartheta = \varphi = 0$, so ist das Ziel erreicht – es wurde in diesem Fall

Ziel-
funktion

ein Testfall gefunden, der das Zustandsübergangsdiagramm auf dem Zielpfad durchläuft.

Kalaji *et al.* erweitern dieses Verfahren durch ein suchbasiertes Analyseverfahren, welches durch Anwendung genetischer Algorithmen einen Satz erreichbarer Ausführungspfade generiert, welche anschließend, dem Verfahren von Lefticaru und Ipate ähnelnd, zur Generierung relevanter Testdaten verwendet werden [73].

2.2.3 KOMMERZIELLE WERKZEUGE

Für die Aufgabe der strukturorientierten Testdatengenerierung für SL/SF Modelle existiert eine Reihe kommerzieller Werkzeuge. Die bekanntesten Vertreter sollen in Kürze vorgestellt werden. Da es sich bei diesen Werkzeugen um kommerzielle Produkte handelt, sind nur relativ wenig Informationen über deren interne Funktionsweise bekannt. Die folgenden Beschreibungen bestehen daher aus Informationen, die entweder aus technischen Dokumenten der Hersteller oder aus anderweitig veröffentlichten Untersuchungen stammen.

REACTIVE SYSTEMS REACTIS

Reactis ist ein von Reactive Systems, Inc. kommerziell zur Verfügung gestelltes Werkzeug zur Testdatengenerierung für SL/SF Modelle [111].

*Funktio-
nalität*

Die verwendete Methode besteht hauptsächlich aus einer Zufalls-suche, bei der die Eingabedaten durch Monte Carlo Methoden bestimmt werden. Dieses Verfahren wurde von Grosu und Smolka erstmals zum Zweck des Model Checkings verwendet [56]. Darüber hinaus wird auch eine als *gesteuerte Simulation* (engl. guided simulation) bezeichnete Methode verwendet. Sims und DuVarney beschreiben, dass diese Methode auf der Interpretation des Modells und dem Verfolgen des internen Datenflusses basiert [119]. An jeder Verzweigung innerhalb des Modells werden die einbezogenen Datenwerte protokolliert. Nach jedem Simulationslauf wird eine Datenflussanalyse auf jene den Kontrollfluss beeinflussenden Werte angewandt, um im folgenden Zyklus neue Systemeingaben zu generieren, welche einen alternativen Ausführungspfad durchlaufen [119]. Es wird demnach eine Kombination aus konkreten

und statischen Ausführungen verwendet. Reactis realisiert daher eine Variante des Concolic Testings.

Reactis unterstützt die Generierung von Testdaten für eine Vielzahl von Überdeckungskriterien, wie z.B. auch MC/DC. Ein generierter Testfall besteht aus einer Matrix. Jede Zeile dieser Matrix entspricht der Sequenz von Datenwerten eines Eingabeports des Modells, jede Spalte repräsentiert hingegen einen Zeitschritt. Nach der Ausführung bzw. Evaluation eines Testfalls wird das Modell in den initialen Zustand zurück versetzt, um einen anderen Testfall ausführen zu können [119].

Testdaten

T-VEC TESTER FOR SIMULINK AND STATEFLOW

T-VEC Tester for Simulink and Stateflow ist ein von T-VEC kommerziell zur Verfügung gestelltes Werkzeug zur Testdatengenerierung für SL/SF Modelle [122].

Vergleichbar mit Reactis werden auch hier Testdaten für eine Reihe unterstützter Überdeckungskriterien generiert. Dazu werden „fortschrittliche Algorithmen“ verwendet [122]. Durch vorherige umfangreiche Modellanalysen sollen bereits vor der Testdatengenerierung eventuelle Modellierungsfehler ausfindig gemacht werden. Dazu zählen beispielsweise widersprüchliche Anweisungen oder Probleme durch sich wechselseitig beeinflussende Merkmale, die zu unerreichbarem Code oder anderen unerwünschten Effekten führen können.

Funktionalität

Alur *et al.* gruppieren T-VEC Tester in die Gruppe jener Werkzeuge zur Testdatengenerierung, welche eine Kombination aus zufallsbasierter Suche und symbolischer Ausführung mit Constraint Solving verwenden [4]. Darüber hinaus ist leider keinerlei Information über die verwendete Methodik verfügbar.

MATHWORKS SIMULINK DESIGN VERIFIER

Der *Simulink Design Verifier* von The MathWorksTM, Inc. ist ein weiteres Werkzeug zur Testdatengenerierung für SL/SF Modelle [127].

Neben der drei unterstützten Überdeckungskriterien DC, CC und MC/DC können vom Benutzer auch eigene Kriterien definiert werden. Das Werkzeug analysiert zur Testdatengenerierung

Funktionalität

die Algorithmen und die Logik des Testobjekts. Dazu wird das *Prover*[®] *Plug-In* [107] von Prover Technology verwendet, welches formale Analysemethoden für SL Modelle und SF Diagramme zur Verfügung stellt.

Laut Alur *et al.* realisiert das Werkzeug ebenfalls eine Kombination aus zufallsbasierter Suche und symbolischer Ausführung mit Constraint Solving [4]. Darüber hinaus ist jedoch keinerlei Information über die verwendete Methodik verfügbar.

2.2.4 GRENZEN EXISTIERENDER ANSÄTZE

Die Einschränkungen und Grenzen der vorgestellten Verfahren wurden in den jeweiligen Beschreibungen bereits angedeutet. Dieser Abschnitt geht genauer auf die erwähnten Einschränkungen ein und fasst diese systematisch zusammen.

Es wurden drei grundlegende Probleme identifiziert, die in den folgenden Abschnitten erläutert werden:

- A. Die verwendete symbolische Ausführung mitsamt des Constraint Solvings weist bei komplexen Testobjekten Schwächen auf.
- B. Die Plausibilität der generierten Testdaten ist nicht gewährleistet.
- C. Das Vorhandensein komplexer Prädikate führt zu einer geringen Effektivität der Verfahren.

EINSCHRÄNKUNGEN SYMBOLISCHER AUSFÜHRUNG

Auf symbolischer Ausführung basierende Verfahren stoßen aufgrund zweier grundlegender Probleme häufig an ihre Grenzen. Zum einen kann ein erfolgreiches Erfassen der notwendigen symbolischen Zustände scheitern und zum anderen haben die heutigen Constraint Solver Probleme, die abgeleiteten Prädikatsysteme zu lösen.

Ausführungsbaum

Das Erfassen der symbolischen Zustände, z.B. in Form eines symbolischen Ausführungsbaums, kann beim Vorhandensein von Schleifen im Programmcode scheitern, z.B. als M-Code eingebettet

in ein SL Modell, oder bei Verwendung dynamisch veränderbarer Elemente an Datenfluss-beeinflussenden Verzweigungen. Das Vorhandensein von Schleifen kann zu einer großen und potentiell unendlichen Anzahl symbolischer Zustände führen, die analysiert werden muss. Somit kann es dazu kommen, dass die symbolische Ausführung nicht terminiert [108].

Das Vorhandensein vieler verzweigender und tief verschachtelter Bedingungen im Testobjekt führt darüber hinaus dazu, dass die Anzahl symbolischer Zustände derart groß wird, dass ein Lösen der resultierenden Constraints nicht praktikabel ist. Die Größe des zu erstellenden symbolischen Ausführungsbaums wächst exponentiell mit der Anzahl der vorhandenen Verzweigungen. Dieses Problem wird auch als *Zustandsraumexplosion* (engl. State Space Explosion) bezeichnet. Erschwerend kommt hinzu, dass bestimmte Zustände eingebetteter Systeme möglicherweise erst nach Durchlaufen einer Vielzahl anderer Zustände erreicht werden können. Die abzuleitenden Testdaten müssen also eine entsprechende Mindestlänge aufweisen. Die Anzahl der symbolischen Zustände wächst jedoch linear mit der Anzahl der berücksichtigten Zeitschritte.

*State Space
Explosion*

Die teilweise für reine Zustandsübergangsdiagramme entwickelten Verfahren sind nicht direkt auf SF Diagramme anwendbar, weil diese potentiell in einem tief verschachtelten SL Block eingebettet sind. Die Eingaben zu jenem SF Diagramm durchlaufen zuvor ein komplexes Netz aus SL Blöcken. Bei diesen SL Blöcken kann es sich auch um nicht-lineare Blöcke handeln. Dies hat nicht-lineare Constraints zur Folge, für deren Lösung keine zufriedenstellenden Methoden existieren [115].

*Nicht-
lineare
Constraints*

Jeder automatische Testdatengenerierungsansatz, der hauptsächlich auf symbolischer Ausführung und Constraint Solving basiert, weist diese Probleme auf. Trotz der vorgestellten Versuche, diese Probleme durch Anwendung von Heuristiken, durch Approximationen oder durch Kombination mit konkreter Ausführung zu lösen, besteht das Hauptproblem nach wie vor in der Skalierbarkeit dieser Verfahren [108].

EINGESCHRÄNKTE PLAUSIBILITÄT DER TESTDATEN

Wie einführend in Kapitel 2.1.4 auf Seite 18 beschrieben, sollen die mit Hilfe des Strukturtests ermittelten Testdaten der Anreicherung

des für funktionale Tests erstellten Testdatensatzes dienen. Dafür müssen die generierten Testdaten den jeweiligen Anforderungen des Testobjekts genügen, da sie ansonsten nicht zu verwenden sind. Im Falle von SL/SF Modellen zählt dazu das Einhalten etwaiger Signalcharakteristika, eventueller minimaler Signallängen oder aber auch das Erfüllen gegebener Constraints.

*Signal-
stetigkeit*

Eine Anforderung eingebetteter Systeme kann die Stetigkeit der dafür generierten Eingabesignale sein, da sich dieses ansonsten aufgrund eventueller intern durchgeführter Plausibilitätsuntersuchungen unterschiedlich verhält. Die häufig angewandte zufällige Generierung von Testdaten oder die unabhängig voneinander durchgeführte Analyse jedes einzelnen Zeitschritts, führt zu einer fehlenden Stetigkeit der Signale. Dies kann somit ein Hindernis für die Generierung plausibler Testdaten darstellen. Als Beispiel sei hier eine Tempomatenfunktion genannt, deren Funktion u.a. von der momentan gefahrenen Geschwindigkeit abhängt. Wenn diese innerhalb kürzester Zeit (z.B. innerhalb einer Millisekunde) von 20 km/h auf 90 km/h springt, wird die Tempomatenfunktion ein Fehlverhalten des Geschwindigkeitssensors registrieren. Derartige unrealistische Fluktuationen müssen daher vermieden werden.

Signallänge

Die Länge der generierten Signale, d.h. die Anzahl der einbezogenen Zeitschritte, wird bei den vorgenannten Verfahren ebenfalls nicht ausreichend berücksichtigt. Das Erreichen eines bestimmten Zustands ist möglicherweise vom vorherigen Durchlaufen einer Vielzahl anderer Zustände abhängig. Dies kann zur Notwendigkeit einer minimalen Signallänge führen. Ein Beispiel ist die Steuerung eines Automatikgetriebes: Bei gegebener Plausibilität der Testdaten, müssen zum Erreichen des höchsten Gangs alle vorherigen Gänge durchgeschaltet und das (virtuelle) Fahrzeug demnach entsprechend lang beschleunigt werden. Sowohl die statischen als auch die dynamischen Verfahren sind in diesem Fall bei separater Betrachtung der einzelnen Zeitschritte nicht mehr praktikabel.

*Signal
Constraints*

Eine weitere wichtige Bedingung für die Plausibilität und Ausführbarkeit der generierten Testdaten kann das Einhalten gegebener Constraints sein. Dazu zählen zum einen elementare Constraints der einzelnen Eingaben, wie beispielsweise einzuhaltende minimale und maximale Datenwerte. Zum anderen zählen dazu allerdings auch Constraints, die das Signalverhalten selbst einschränken, wie z.B. eine maximal oder minimal einzuhaltende

Steigung des Signals. Ebenso können komplexere Constraints und auch wechselseitige Abhängigkeiten mehrerer Signale gegeben sein.

Keine der in Kapitel 2.2 auf Seite 23 vorgestellten Verfahren berücksichtigt die genannten Anforderungen eingebetteter Systeme an die in Form von Signalen zu generierenden Testdaten.

EINGESCHRÄNKTE EFFEKTIVITÄT BEI KOMPLEXEN PRÄDIKATEN

Die vorgestellten Verfahren stoßen insbesondere bei komplexen Prädikaten an ihre Grenzen. Ein Prädikat wird als komplex angesehen, wenn es aus mehreren nicht-trivialen Teilbedingungen besteht, d.h. aus jenen Bedingungen, deren erfüllende Testdaten im gesamten Suchraum einen nur sehr kleinen Bereich einnehmen, und daher sehr selten zufällig gefunden werden. Triviale Bedingungen werden demgegenüber von der Mehrheit der Testdaten erfüllt. Im Suchraum der 16-Bit-Integer-Zahlen gilt die Bedingung $x > 0.0$ beispielsweise als trivial, da die Hälfte aller Werte des Suchraums (50%) die Bedingung erfüllen. Im Vergleich dazu gilt die Bedingung $x == 0.0$ als nicht-trivial, da lediglich ein Wert des gesamten Suchraums (0.0015%) diese Bedingung erfüllt.

Komplexe Prädikate sind eine große Herausforderung für Verfahren basierend auf symbolischer Ausführung und Constraint Solving. Bei den statischen Verfahren wird das komplexe Prädikat Teil des zu lösenden Prädikatensystems. Der verwendete Constraint Solver muss dann eine Lösung für diese Prädikaten-system und somit auch für das komplexe Prädikat berechnen. Dies führt im besten Fall zu einem deutlich erhöhten Rechenaufwand; Wahrscheinlicher ist, dass mit den heutigen Constraint Solvern keine Lösung gefunden werden kann [108]. Dies ist vor allem der Fall, wenn die zu lösenden Prädikate nicht-linearer Natur sind [115].

*Probleme
statischer
Verfahren*

Dynamische Verfahren eignen sich grundsätzlich besonders gut zur Lösung komplexer Prädikate. Dies gilt jedoch nur, wenn eine für das jeweilige Prädikat geeignete Zielfunktion definiert wurde. Ist dies nicht der Fall, so wird die Suche nicht ausreichend geleitet. Die Zielfunktionslandschaft weist in diesem Fall Plateaus auf, die keinerlei Aufschluss darüber geben, in welcher Richtung bessere Lösungen zu erwarten sind. Die Suche degeneriert dann zu einer ungesteuerten Zufallssuche. Ein Beispiel dafür ist die von

*Probleme
dynam.
Verfahren*

Norbert Oster *et al.* vorgeschlagene Zielfunktion für die Testdatengenerierung aus UML Modellen [102]. Die Zielfunktion wird ausschließlich basierend auf der erreichten Modellüberdeckung berechnet und bietet somit keinerlei Steuerung bzgl. der dazu notwendigen Erfüllung interner evtl. komplexer Prädikate.

2.3 EVOLUTIONÄRE ALGORITHMEN

Metaheuristiken

Immer dann, wenn analytische Verfahren nicht auf ein gegebenes Problem anwendbar sind oder deren Anwendung zu aufwändig ist, bietet sich der Einsatz suchbasierter Verfahren an. Alle NP-schweren Probleme, d.h. jene, welche nicht-deterministisch in polynomieller Zeit entscheidbar sind, zählen zu diesen Problemen. Zur Lösung dieser Probleme können zwar analytische Verfahren existieren, deren Anwendung ist allerdings aufgrund der exponentiell steigenden Komplexität nicht praktikabel [46]. Praktikabler könnte in diesen Fällen die Anwendung einer Metaheuristik sein. Metaheuristiken sind Algorithmen zur näherungsweise Lösung kombinatorischer Optimierungsprobleme und sind charakterisiert durch eine abstrakte, auf eine Vielzahl unterschiedlicher Optimierungsprobleme anwendbare Folge von Schritten.

Einsatzbereiche

Das Verfahren der evolutionären Algorithmen ist eine solche Metaheuristik und ist besonders geeignet zur Lösung NP-schwerer und komplexer Optimierungsprobleme hoher Dimensionalität, welche potentiell einen nicht-linearen, multimodalen und diskontinuierlichen Suchraum aufweisen [41]. Im Gegensatz zu üblichen Optimierungsverfahren werden keine expliziten Annahmen über das zu lösende Probleme getroffen, so werden beispielsweise keinerlei Gradienteninformationen benötigt. Evolutionäre Algorithmen realisieren eine Black-Box Optimierung, d.h. es wird kein direkter Zugriff auf die Problemistanz benötigt; als Basis für die Steuerung der Suche dient lediglich die Auswertung der erstellten Lösungsvorschläge.

Suchbasierte Testverfahren beinhalten eben solche Problemtypen, wodurch die Anwendung evolutionärer Algorithmen begründet ist. Dieses Unterkapitel beleuchtet die Prinzipien evolutionärer Algorithmen, um die Verständlichkeit des Optimierungsprozesses des suchbasierten Strukturtests zu erhöhen. In Abschnitt 2.3.1 wird zunächst auf die Eigenschaften allgemeiner evolutionärer

Algorithmen und deren Anwendung eingegangen. Abschnitt 2.3.2 erläutert mit der genetischen Programmierung eine besondere Variante evolutionärer Algorithmen. Auf dieser liegt ein besonderes Augenmerk, da sie im suchbasierten Modell-Strukturtest verwendet wird (vgl. Teil ii auf Seite 67).

2.3.1 PRINZIPIEN EVOLUTIONÄRER ALGORITHMEN

Der Begriff *Evolutionäre Algorithmen* umfasst alle Optimierungsverfahren, welche auf Mechanismen der biologischen Evolution basieren. Sie zeichnen sich sowohl durch die Verwendung einer Population von potentiellen Lösungen und der damit verbundenen Parallelität der Suche, als auch durch die verwendeten probabilistischen Übergangsregeln aus. Des Weiteren sind Evolutionäre Algorithmen häufig von biologischen Paradigmen, wie beispielsweise bei Tieren beobachteten Verhaltensweisen oder der Genetik, inspiriert.

Natur als Vorbild

Im Vergleich zu traditionellen Such- und Optimierungsverfahren, bei denen die Suche inkrementell von einem Punkt im Suchraum zum anderen verläuft, starten Evolutionäre Algorithmen die Suche typischerweise mit einer Population von Individuen, welche ähnlich einer gerichteten Zufallssuche den Suchraum erkunden. Es werden keine zusätzlichen Informationen über die zu optimierende Funktion benötigt, wie z.B. Funktionsableitungen, wie sie beispielsweise bei Gradientenverfahren berechnet werden müssen. Evolutionäre Algorithmen bewerten die Güte, auch Fitness genannt, ihrer Individuen (Positionen im Suchraum) nach dem Wert der zu optimierenden Funktion. Die Fitness eines Individuums entscheidet, wie groß dessen Einfluss auf spätere Generationen ist. Dadurch wird die Suche auf Regionen im Suchraum gerichtet, von denen man sich eine höhere Wahrscheinlichkeit für gute Fitnesswerte erhofft.

Charakteristika

Das Feld der Evolutionären Algorithmen lässt sich nach Kennedy und Eberhart [74] in die folgenden vier sich unabhängig voneinander entwickelten Strömungen einteilen:

- Genetische Algorithmen (Holland [63, 64])
- Genetische Programmierung (Koza [79])
- Evolutionsstrategien (Rechenberg [113])

- Evolutionäre Programmierung (Fogel *et al.* [42])

Diese vier Strömungen unterscheiden sich hauptsächlich in der Art der internen Repräsentation der Lösungsvorschläge des untersuchten Optimierungsproblems und der typischerweise angewandten Selektions- und Variationsoperatoren. Im Folgenden wird zunächst genauer auf evolutionäre Algorithmen im Allgemeinen eingegangen, die folgenden zwei Abschnitte stellen genetische Algorithmen und die genetische Programmierung im Speziellen vor. Auf evolutionäre Programmierung und Evolutionsstrategien wird nicht näher eingegangen, da diese im Rahmen dieser Arbeit nicht verwendet werden.

Terminologie

Bei den evolutionären Algorithmen handelt es sich um ein paralleles Optimierungsverfahren, d.h. es werden gleichzeitig mehrere Lösungsvorschläge für das zugrundeliegende Optimierungsproblem bearbeitet. Dadurch soll die Wahrscheinlichkeit der Konvergenz in lokale Optima verringert werden. Ein Satz dieser möglichen Lösungsvorschläge wird als *Population* P bezeichnet. Ein einzelner Lösungsvorschlag aus P ist ein *Individuum* i . Jedes Individuum $i \in P$ besteht aus mindestens einem *Gen* g , welches jeweils einen bestimmten Aspekt des Lösungsvorschlags modelliert. Die Anzahl der Gene wird auch als *Individuenlänge* bezeichnet. Bei komplexeren Datenstrukturen können die Gene noch in einzelne Chromosome gruppiert sein (vgl. Abschnitt 2.3.2 auf Seite 57). Ein Individuum besteht in diesem Fall aus mehreren *Chromosomen* c ($c \in i$), ein Chromosom wiederum besteht aus mehreren Genen ($g \in c$). Die einzelnen Individuen werden bezüglich ihrer Fähigkeit, das zugrundeliegende Optimierungsproblem zu lösen, bewertet. Diese Bewertung wird als *Fitness* des jeweiligen Individuums bezeichnet und spielt eine große Rolle im Optimierungsprozess: Unter Anwendung des Prinzips der natürlichen Selektion⁶ werden iterativ erfolgversprechende Individuen selektiert, welche unter Verwendung von Variationsoperatoren (Rekombination und Mutation) zur Erzeugung neuer Individuen verwendet werden. Jede dieser Iterationen wird als eine *Generation* bezeichnet. Es werden so lange neue Generationen erstellt, bis das Optimierungsziel erreicht ist oder ein anderes Abbruchkriterium eintritt.

Anwendung

Abb. 12 illustriert die allgemeine Anwendung eines evolutionären Algorithmus'. Der evolutionäre Algorithmus benötigt zur Ge-

⁶ Die natürliche Selektion beschreibt das wahrscheinlichere Überleben von Individuen mit höherer Fitness gegenüber Individuen mit geringerer Fitness.

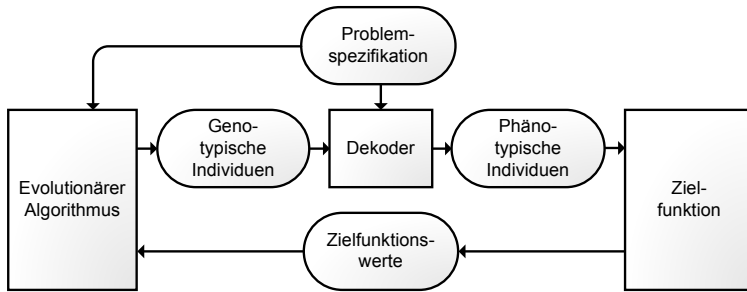


Abb. 12: Anwendung eines evolutionären Algorithmus': Zusammenspiel des genotypischen und des phänotypischen Suchraums (nach Wappler [145]).

nerierung sinnvoller Lösungsvorschläge eine *Problemspezifikation*, welche Informationen über die Natur der zu generierenden Lösungsvorschläge enthält. Aus der Problemspezifikation ergibt sich die *Repräsentation*, welche wiederum beschreibt, wie eine Lösung im phänotypischen Suchraum als eine potentielle Lösung im genotypischen Suchraum kodiert ist. Der *phänotypische Suchraum* beschreibt die Menge aller möglichen Lösungsvorschläge des Optimierungsproblems während der *genotypische Suchraum* die Menge aller Instanziierungen der verwendeten Repräsentation darstellt [145]. Der *Dekoder* interpretiert ein genotypisches Individuum als eine Lösung des Optimierungsproblems; er wandelt die von den evolutionären Algorithmen verwendeten Datenstrukturen in evaluierbare Lösungsvorschläge für das Optimierungsproblem um.

Ein Beispiel dafür ist das in Kapitel 3 auf Seite 67 beschriebene Paradigma der Signalgenerierung im Rahmen des suchbasierten Modelltests. Die Problemspezifikation enthält in diesem Fall für jedes zu generierende Signal spezifische Parameter, wie beispielsweise die maximalen oder minimalen Amplitudenwerte oder evtl. vorhandene Signal Constraints. Je nach verwendetem Verfahren, handelt es sich bei den genotypischen Individuen entweder um einen Parametersatz oder eine variable Datenstruktur, auf der die evolutionären Algorithmen operieren. Bei dem Dekoder handelt es sich in diesem Beispiel um den Signalgenerator, der aus diesen Daten der Problemspezifikation entsprechende Signale erzeugt. Die evolutionären Algorithmen operieren demnach nicht auf den Signalen selbst, sondern auf den Parametern, aus denen

Beispiel

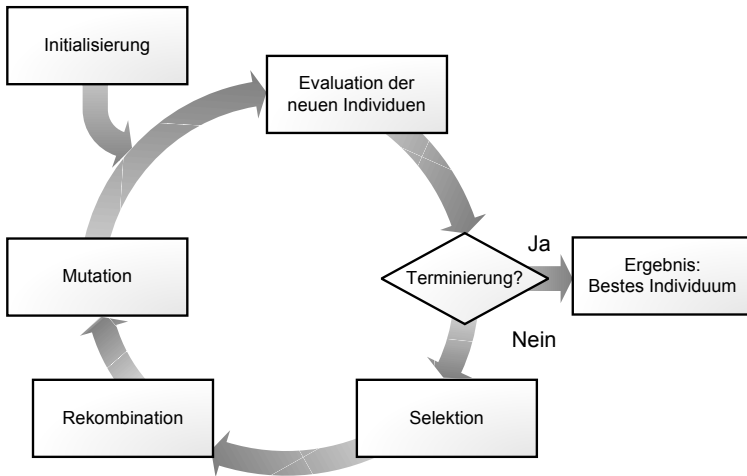


Abb. 13: Ablauf eines evolutionären Algorithmus'

sich die Signale ergeben. Zur Bewertung der Individuen wird eine *Zielfunktion* benötigt, welche für jedes Individuum eine Bewertung (in Form einer reellen Zahl) liefert, inwiefern dieses zur Lösung des zugrundeliegenden Optimierungsproblems geeignet ist. Ungeeignete Individuen erhalten einen schlechten Zielfunktionswert, während vielversprechende Individuen einen guten Zielfunktionswert erhalten.

Ablauf der
Optimie-
rung

In Abb. 13 ist der Ablauf eines einfachen evolutionären Algorithmus' dargestellt. Am Anfang des Optimierungsprozesses steht die *Initialisierung*, welche die üblicherweise zufällige Generierung einer initialen Population beinhaltet. Für jedes Individuum dieser Population wird ein Zielfunktionswert bestimmt (*Evaluation*). Ist das ausgewählte *Abbruchkriterium* erfüllt, so kann das Individuum mit der besten Fitness als Optimierungsergebnis zurückgegeben werden. Falls nicht, werden zyklisch neue Generationen generiert, bis das Abbruchkriterium erfüllt ist. Jede Generation entsteht unter Anwendung der evolutionären Operatoren *Selektion*, *Rekombination* und *Mutation*. Durch die Selektion wird je nach verwendeter Selektionsart ein Satz von Individuen in Abhängigkeit ihrer Fitness, in diesem Zusammenhang *Eltern* genannt, für die Produktion von neuen Individuen, den *Nachkommen*, ausgewählt. Die Eltern werden dann mit Hilfe der Rekombination zur Erstellung ihrer Nachkommen verwendet, welche anschließend

durch die Mutation, zufällig und gemäß einer zuvor definierten Wahrscheinlichkeit, modifiziert werden. Anschließend werden die Nachkommen in die Population eingefügt und ersetzen damit je nach Operator beispielsweise die im Sinne der Zielfunktion schwächsten Individuen (*Wiedereinfügen*). Mit der Bewertung der neuen Individuen ist die Generierung der neuen Generation abgeschlossen, von der eine – im Vergleich zur vorherigen Generation – bessere oder mindestens gleichwertige Gesamtfitness erhofft wird.

Der folgende Abschnitt beschreibt die einzelnen Operatoren im Detail. Es werden jeweils unterschiedliche Herangehensweisen vorgestellt. Neben den gängigen Verfahren werden vor allem die für die Anwendung im suchbasierten Modell-Strukturtest (siehe Teil ii auf Seite 67) verwendeten Verfahren genauer beschrieben.

OPERATOREN EVOLUTIONÄRER ALGORITHMEN

Die Initialisierung erstellt die initiale Population P_0 . Falls kein Vorwissen über die Lage der zu suchenden Lösung vorliegt, wird die Startpopulation zufällig im Definitionsbereich der Variablen initialisiert. Falls Vorwissen vorhanden ist, beispielsweise aufgrund vorheriger Optimierungsläufe, kann diese Information in die Optimierung einfließen. Dies geschieht entweder durch die Verteilung der Startpopulation in der Umgebung der erwarteten Lösungsposition oder durch einfaches Einfügen der vorherigen Ergebnisse als einzelne Individuen. Dieses Vorgehen wird als *Seeding* bezeichnet.

Initialisierung

Während der Evaluation wird jedem Individuum ein Zielfunktionswert zugewiesen, d.h. jedes Individuum wird mit Hinblick auf seine Eignung, das zugrundeliegende Optimierungsproblem zu lösen, bewertet. Dazu werden die Individuen nacheinander an die Zielfunktion übergeben, welche die Bewertung für jedes Individuum zurückgibt. Je nach Optimierungsziel kann sich die Zielfunktion stark unterscheiden. Liegt das Ziel beispielsweise in der Optimierung einer mathematischen Benchmarkfunktion, so handelt es sich bei der Zielfunktion direkt um diese mathematische Funktion. Wird der evolutionäre Algorithmus demgegenüber im Rahmen eines dynamischen Testverfahrens eingesetzt, wird das zu bewertende Individuum in konkrete Testdaten umgewandelt und zur Ausführung des Testobjekts verwendet. Basierend

Evaluation

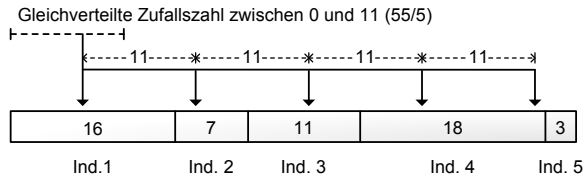


Abb. 14: Exemplarische Illustration des Stochastic Universal Sampling. Die gleichmäßig verteilten Zeiger definieren die zu selektierenden Individuen.

auf dem Verhalten des Testobjekts wird eine Bewertung des Individuums hinsichtlich des Erreichens eines Testziels abgeleitet, welche anschließend in Form eines numerischen Wertes als Ergebnis der Zielfunktion zurückgegeben wird.

Selektion

Eine Selektion wird immer dann durchgeführt, wenn bestimmte Individuen unter Berücksichtigung ihrer Fitness aus einer Population ausgewählt werden müssen. Dabei ist zu berücksichtigen, dass die verwendete Selektionsstrategie nicht nur jene Individuen mit der größten Fitness auswählen, sondern mit einer bestimmten Wahrscheinlichkeit ebenfalls Individuen mit niedriger Fitness berücksichtigen. Anderenfalls würde sich die Fähigkeit des Verfahrens verringern, lokale Optima zu verlassen. Es existieren unterschiedliche Konzepte, wobei im Folgenden genauer auf die beiden Strategien Stochastic Universal Sampling und Turniersélection eingegangen wird, da diese im Rahmen dieser Arbeit zur Anwendung kommen. Das *Stochastic Universal Sampling* [7] ordnet allen Individuen des Selektionspools⁷, entsprechend ihrer Fitness, Abschnitte auf einer Linie zugeordnet, deren Länge der gesamten Fitness des Selektionspools entspricht. Individuen mit hoher Fitness nehmen mehr Platz in Anspruch als Individuen mit niedriger Fitness. Es werden wie in Abbildung Abb. 14 illustriert, entsprechend der Anzahl der zu selektierenden Individuen Zeiger mit konstanten Abständen auf der Linie verteilt. Diese Abstände ergeben sich aus der Division der Linienlänge und der Anzahl der zu selektierenden Individuen. Die Position des ersten dieser Zeiger wird durch das Ziehen einer gleichverteilten Zufallsvariable im Bereich zwischen 0 und dem definierten Zeigerabstand bestimmt. Alle Individuen, auf dessen Linienbereich

⁷ Der Selektionspool ist eine Menge von Individuen, für die eine Selektion durchgeführt werden soll. Dies kann sowohl die gesamte Population als auch eine Teilmenge (Subpopulation) davon sein.

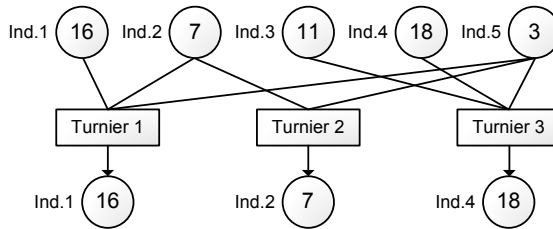


Abb. 15: Exemplarische Illustration der Turnierselektion. Verschiedene Individuen treten in Turnieren gegeneinander an: Jenes mit der höchsten Fitness gewinnt.

ein Zeiger zeigt, werden selektiert. Vorteile dieses Verfahrens sind zum einen die hohe Geschwindigkeit, da nur eine einzige Zufallszahl generiert werden muss, und zum anderen die Vermeidung der möglichen, ausschließlichen Selektion von Individuen niedriger Fitness. Anzumerken ist, dass Individuen mehrfach selektiert werden können, falls das Individuum eine entsprechend hohe Fitness aufweist. Ein weiteres Selektionsverfahren ist die *Turnierselektion* (engl. tournament selection) [51]. Es bestimmt zufällig und gleichverteilt mehrere Individuen des Selektionspools und vergleicht deren Fitnesswerte. Das jeweils beste Individuum, d.h. jenes mit der höchsten Fitness, wird ausgewählt. Derartige Turniere werden so oft ausgeführt, bis die gewünschte Anzahl an zu selektierenden Individuen erreicht ist. Abb. 15 zeigt ein Beispiel mit drei Turnieren zwischen verschiedenen Individuen eines Selektionspools. Die Anzahl der für ein Turnier zu berücksichtigenden Individuen bestimmt den *Selektionsdruck*⁸. Je geringer der Selektionsdruck für die Individuen einer Population ist, desto höher fällt die Diversität der von ihnen durchgeführten Suche aus.

Die Rekombination ist einer der wichtigsten Operatoren evolutionärer Algorithmen und basiert auf der Metapher der geschlechtlichen Fortpflanzung. Durch den Austausch und die Kombination von Informationen werden aus mindestens zwei zuvor selektierten Individuen Nachkommen erzeugt. Wie viele Nachkommen innerhalb einer Generation erzeugt werden sollen, wird bestimmt durch die *Generationslücke* (engl. generation gap), welche den

*Rekombi-
nation*

⁸ Der Selektionsdruck ist die Notwendigkeit eines Individuums, sich seiner Umgebung anzupassen. Er ist ein Maß für die Bevorzugung der besten gegenüber den schlechteren Individuen.

Anteil der Nachkommen bezogen auf die Gesamtpopulation angibt. Zusätzlich gibt die *Rekombinationswahrscheinlichkeit* P_R die Wahrscheinlichkeit an, mit der eine Rekombination überhaupt durchgeführt wird. Es existieren unterschiedliche Verfahren, die je nach Datenrepräsentation und Anwendungsgebiet verwendet werden können. Häufig verwendete Rekombinationsstrategien werden für die genetische Programmierung in Abschnitt 2.3.2 beschrieben.

Mutation

Die Mutation wird in der Regel nach der Rekombination mit einer geringen Wahrscheinlichkeit auf die neu generierten Individuen angewandt. Die Individuen werden geringfügig modifiziert, um die Diversität der gesamten Population zu erhalten und der vorzeitigen Konvergenz in lokale Optima entgegenzuwirken, da durch die Mutation eine zu große Ähnlichkeit der Individuen verhindert wird. Als *Mutationswahrscheinlichkeit* P_M wird die Wahrscheinlichkeit bezeichnet, mit der der Mutationsoperator angewandt wird. Es existieren Mutationswahrscheinlichkeiten für Individuen P_{M_i} , für die enthaltenen Chromosomen P_{M_c} und für die wiederum enthaltenen Gene P_{M_g} . Das Ausmaß der Modifikationen bezeichnet man als *Mutationsschritt* (engl. mutation range) und deren Häufigkeit als *Mutationsrate* (engl. mutation rate). Es hat sich gezeigt, dass eine Adaption dieser Parameter während der Suche von Vorteil ist [16]: Da der evolutionäre Algorithmus zu Beginn der Optimierung weite Bereiche des Suchraums erkunden und erst bei voranschreitender Suche detailliertere Auflösungen verwenden sollte, ist eine proportional zum Optimierungsfortschritt zu haltende Mutationsschrittweite vorteilhaft [16, 38].

Wiedereinfügen

Das Wiedereinfügen (engl. reinsertion) bestimmt, welche der durch die Rekombination und anschließende Mutation erzeugten Individuen in die ursprüngliche Population aufgenommen werden sollen. Der Parameter *Wiedereinfügerate* (engl. reinsertion rate) bestimmt den maximalen Anteil der Individuen der ursprünglichen Population, die durch Nachkommen ersetzt werden sollen. Eine Wiedereinfügerate von 1.0 bedeutet, dass alle Individuen durch Nachkommen ersetzt werden. Ist die Anzahl der zu ersetzenden Individuen kleiner als die in Abhängigkeit der Generationslücke erstellten Nachkommen, so muss entschieden werden, welche der Nachkommen eingefügt werden sollen. Dies kann sowohl von deren erreichten Fitness als auch von deren

Alter⁹ abhängen. Abhängig von der gewählten Generationslücke und der Wiedereinfügerate lassen sich mehrere Strategien unterscheiden. In dieser Arbeit wird das *elitäre Wiedereinfügen* (engl. *elitest reinsertion*) verwendet. Diese Variante des Wiedereinfügens ersetzt die jeweils schlechtesten Individuen. Ist die Anzahl der erzeugten Nachkommen größer als die Anzahl der einzufügenden Individuen (Generationslücke > Wiedereinfügerate), so muss entschieden werden, welche der erzeugten Nachkommen eingefügt werden. Es werden üblicherweise die bezüglich ihrer Fitness besten Individuen ausgewählt.

Es existiert eine Vielzahl von in diesem Kontext anwendbaren *Abbruchkriterien*, wie z.B. das Erreichen eines bestimmten Fitnesswertes oder auch die allgemeine Konvergenz der Population. Diese Kriterien besitzen jedoch keinen garantierten Abbruch, wodurch es zu keinem Ende der Optimierung kommt, falls der Algorithmus die Kriterien nicht erreicht. Derartige Kriterien müssen aus diesem Grund mit garantiert endenden Abbruchkriterien kombiniert werden. Beispiele dafür sind das Erreichen einer maximalen Anzahl von Generationen oder eine maximal zulässige Rechenzeit.

*Abbruch-
kriterium*

2.3.2 GENETISCHE PROGRAMMIERUNG

Bei der genetischen Programmierung (GP) handelt es sich um eine spezielle Variante evolutionärer Algorithmen, die suchbasiert Computerprogramme zur Erfüllung einer gegebenen Aufgabe erstellt. Die genetische Programmierung kann Datenstrukturen dynamischer Länge erstellen und optimieren. Diese dynamischen Datenstrukturen repräsentieren das zu entwickelnde Programm, üblicherweise bestehend aus hierarchisch aufgebauten Programmanweisungen.

Vorreiter auf diesem Gebiet ist Koza, der in den späten 80er Jahren die genetische Programmierung zur Erzeugung einfacher in Baumstruktur vorliegender Programme nutzte [79]. Die Verwendung von Baumstrukturen für die Repräsentation von prozeduralem Programmcode wurde erstmalig von Cramer vorgeschlagen [33]. Neben Baumstrukturen können als Repräsentation bei den genotypischen Individuen alternativ auch lineare Datenstrukturen

Historie

⁹ Das Alter eines Individuums entspricht hier der Anzahl der bereits überlebten Generationen.

oder Graphenstrukturen zum Einsatz kommen [10]. Auf die Verwendung linearer Datenstrukturen im Rahmen der genetischen Programmierung – auch als *lineare genetische Programmierung* bekannt – wird im Folgenden genauer eingegangen, da deren Prinzipien und die angewandten Variationsoperatoren für diese Arbeit von besonderem Interesse sind.

LINEARE GENETISCHE PROGRAMMIERUNG

Im Unterschied zur üblicherweise auf Baumstrukturen basierenden klassischen genetischen Programmierung wird bei der linearen genetischen Programmierung (LGP) auf lineare Datenstrukturen zurückgegriffen [22, 105]. Ebenfalls zur suchbasierten Erstellung von Computerprogrammen eingesetzt, besteht ein genotypisches Individuum aus einer linearen Aneinanderreihung von Anweisungen. Dies entspricht im Vergleich zu einer Baumstruktur eher der imperativen Natur der Programmausführung herkömmlicher anweisungsbasierter Programmiersprachen. Im Rahmen der genetischen Programmierung wurden lineare Datenstrukturen erstmalig von Cramer verwendet [33]. Ein allgemeiner Ansatz der LGP wurde von Banzhaf vorgestellt [9], während Nordin ein Verfahren zur direkten Optimierung des Maschinencodes unter Verwendung einer linearen Repräsentation vorstellte [103].

Kodierung

Bei der LGP ist eine Sequenz von Programmanweisungen (das eigentliche Programm) in Form einer linearen Datenstruktur kodiert. Jede dieser Programmanweisungen besteht aus einem Operator und einer Liste von Registern auf denen operiert wird. So lassen sich die meisten Anweisungen mit Hilfe von vier numerischen Werten darstellen: Ein Wert als Identifikator für den zu verwendenden Operator und drei Werte für die Identifikatoren des Zielregisters und der beiden Register der Operanden. Diese vier Werte (8-Bit-Integer) können dann zu einem ganzzahligen Wert (32-Bit-Integer) zusammengefasst werden, d.h. alle Programmanweisungen sind als einzelne Zahlen in einer linearen Datenstruktur kodiert. Die linearen Datenstrukturen, d.h. das gesamte kodierte Programm, wird als *Chromosom* und eine Programmanweisung als *Gen* bezeichnet. Abb. 16 zeigt ein beispielhaftes Individuum (16843012, 33686020, 50528514). Die Anweisung $x = x + 1$; wird als das Quadrupel aus den 8-Bit-Integer-Zahlen 1 (Identifikator des Additions-Operators), 1 (Identifikator für das Zielregister der Variablen x), 1 (Identifikator für das Register des

```

int f(int a, int b)
{

```

1	x = x + 1;	00000001	00000001	00000001	00000100	= 16843012
2	y = y - 1;	00000010	00000010	00000010	00000100	= 33686020
3	return x * y;	00000011	00000011	00000001	00000001	= 50528514

```

}

```

Abb. 16: Kodierung der Programmanweisungen im Rahmen der LGP nach Brameier: Jede Anweisung ist durch einen ganzzahligen Wert (32-Bit-Integer) repräsentiert.

ersten Operators) und 4 (Identifikator für das Register der Konstanten 1) in der 32-Bit-Zahl 16843012 zusammengefasst. Alle folgenden Anweisungen werden analog dazu kodiert.

Der in dieser Arbeit entwickelte Signalgenerierungsansatz basiert auf einer linearen Datenstruktur und ist daher gut für die Anwendung der Prinzipien der LGP geeignet (vgl. Kapitel 3 auf Seite 67). Im Folgenden werden ausgewählte, im Rahmen der LGP anwendbare Initialisierungs- und Variationsoperatoren vorgestellt.

INITIALISIERUNGSSTRATEGIEN

Bei der Erstellung der initialen Population P_0 muss bei der LGP auf die Einhaltung minimaler und maximaler Programmlängen (Anzahl der Programmanweisungen) geachtet werden. Die Anzahl der zu erstellenden Programmanweisungen werden je Individuum innerhalb dieser oberen und unteren Schranke gleichverteilt zufällig bestimmt. Jede Programmanweisung wird dabei ebenfalls zufällig aus dem zur Verfügung stehenden Sätzen an Operatoren und Registern bestimmt.

REKOMBINATIONSTRATEGIEN

Alle Rekombinationsstrategien basieren auf dem Austausch von Genen zwischen zwei ausgewählten Individuen. Eine dieser Strategien ist die *homologe Rekombination*, welche die Gene zweier Elternindividuen homolog kombiniert [81, 104]. Die Strategie ist dahingehend homolog, dass sie, wie in Abb. 17a illustriert, lediglich Gene in einem zufällig bestimmten für beide Elternindividuen gültigen Bereich austauscht. Für diesen Zweck sind die Schnittpunkte

Varianten



Abb. 17: Rekombinationsoperatoren der linearen genetischen Programmierung unter Verwendung zufällig bestimmter Schnittpunkte: (a) Homologe Rekombination, (b) Zwei-Punkt-Rekombination, (c) Ein-Punkt-Rekombination.

sp_1 und sp_2 für beide Elternindividuen identisch und auch die Länge des Nachkommens l_n stimmt dadurch mit der Länge des ersten Elters l_{e_1} überein. Eine weitere Rekombinationsstrategie ist die in Abb. 17b illustrierte *Zwei-Punkt-Rekombination*, welche der homologen Rekombination ähnelt. Der einzige Unterschied besteht darin, dass sich die beiden Bereiche, aus denen die Gene ausgetauscht werden, für beide Elternindividuen unterscheiden können. Für beide Elternindividuen werden hierbei zufällig zwei Schnittpunkte bestimmt: sp_{1,e_1} und sp_{2,e_1} definieren den Schnittbereich des ersten Elters e_1 , während sp_{1,e_2} und sp_{2,e_2} jenen des

zweiten Elters e_2 definiert. Basierend auf der Länge der jeweils bestimmten Bereiche, kann die Länge des Nachkommens stark variieren. Bei der *Ein-Punkt-Rekombination* wird für beide Elternindividuen jeweils zufällig ein Schnittpunkt bestimmt (sp_1 und sp_2), ab denen die jeweils folgenden Gene zur Erstellung eines Nachkommens ausgetauscht werden (siehe Abb. 17c). Wie bei der *Zwei-Punkt-Rekombination* kann die Länge des Nachkommens basierend auf den gewählten Schnittpunkten stark variieren.

Für die zufällige Bestimmung der Schnittpunkte muss bei allen Rekombinationsstrategien auf die maximal oder minimal zulässige Länge der zu erstellenden Nachkommen geachtet werden ($l_{\min}$ bzw. $l_{\max}$). Dies gilt insbesondere für die beiden zuletzt vorgestellten längenvariierenden Strategien. Dies setzt voraus, dass die Elternindividuen diese Schranken bereits berücksichtigen: $l_{\min} \leq l_{e_1} \leq l_{\max} \wedge l_{\min} \leq l_{e_2} \leq l_{\max}$. Bei der homologen Rekombination muss lediglich sichergestellt werden, dass die Schnittpunkte für beide Elternindividuen gültig sind, d.h. $sp_1 < sp_2 \wedge sp_2 < \min(l_{e_1}, l_{e_2})$. Bei der *Zwei-Punkt-Rekombination* muss die mögliche Über- oder Unterschreitung der maximalen bzw. minimalen Länge des Nachkommens verhindert werden: $l_n = l_{e_1} + sp_{2,e_1} - sp_{1,e_1} - sp_{2,e_2} + sp_{1,e_2}$ (es gilt $sp_{1,e_1} < sp_{2,e_1} \wedge sp_{1,e_2} < sp_{2,e_2}$). Für die *Ein-Punkt-Rekombination* müssen ebenfalls die Längenbeschränkungen des Nachkommens beachtet werden: $l_n = sp_{e_1} + l_{e_2} - sp_{e_2}$.

Nebenbedingungen

MUTATIONSSTRATEGIEN

Nach Brameier wird bei der linearen genetischen Programmierung in Makro- und Mikromutationen unterschieden [22]. *Makromutation* beschreibt die Veränderung des Individuums durch Entfernen oder Hinzufügen von Genen, d.h. durch das Verändern der Individuenlänge. Demgegenüber beschreibt die *Mikromutation* jene Veränderungen, die ein oder mehrere Gene selbst beeinflussen; die Individuenlänge wird nicht verändert.

Abb. 18 illustriert drei unterschiedliche Mutationsoperatoren. Die *erweiternde Mutation* zählt zu den Techniken der Makromutation, da sie das zu mutierende Individuum durch Anhängen einer zufälligen (kleinen) Anzahl c neu erstellter Gene erweitert (vgl. Abb. 18a auf der nächsten Seite). Die in Abb. 18b beispielhaft illustrierte *reduzierende Mutation* ist ebenfalls eine Makromutationstechnik; die zu mutierenden Individuen werden um einige Ge-

Varianten

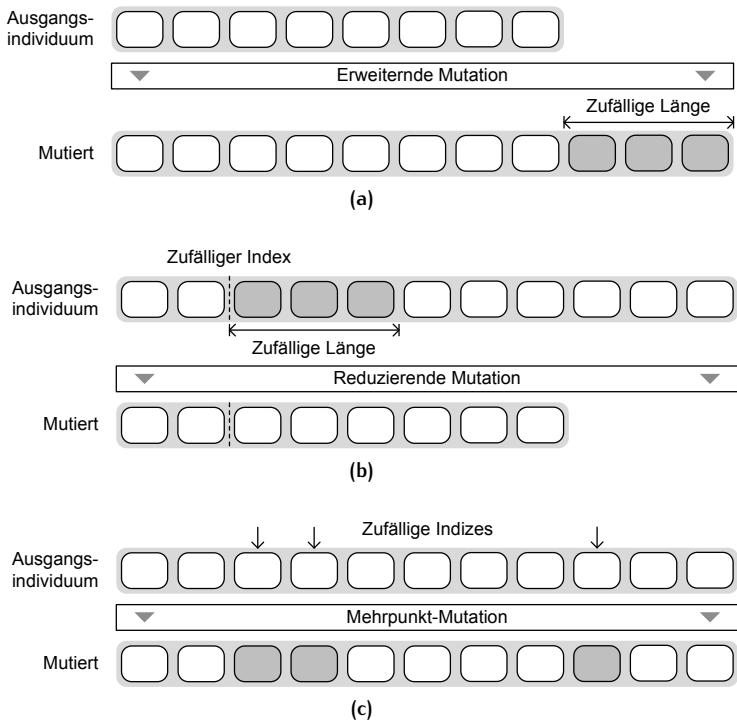


Abb. 18: Mutationsoperatoren für die lineare genetische Programmierung: (a) Erweiternde Mutation, (b) Reduzierende Mutation, (c) Mehrpunkt-Mutation.

ne reduziert. Welche der enthaltenen Gene aus dem Individuum entfernt werden, wird bestimmt durch einen zufällig gewählten Startindex i und durch die Anzahl c der zu entfernenden Gene. Zu den Techniken der Mikromutation zählt die in Abb. 18c dargestellte *Mehrpunkt-Mutation*. Diese wählt zufällig eine kleine Anzahl von Genen aus, welche dann geringfügig verändert werden. Diese Veränderungen hängen stark von der jeweils verwendeten Kodierung der Individuen bzw. der Gene ab. Handelt es sich bei den Genen beispielsweise wie zuvor beschrieben um einzelne Programmanweisungen, so kann eine dieser Mutationen darin bestehen, den Index der jeweiligen Anweisung oder die Indizes der verwendeten Register zu verändern.

Nebenbedingungen

Die Einhaltung der minimal und maximal zulässigen Individuenlänge $l_{\min}$ und $l_{\max}$ ist bei den beiden zuerst genannten Mutati-

onstechniken besonders wichtig. Bei der erweiternden Mutation darf die zulässige Maximallänge des mutierten Individuums nicht überschritten werden; d.h. die Anzahl der neu zu erstellenden und anzuhängenden Gene muss zwischen 1 und der Differenz zwischen $l_{\max}$ und der Länge des Ausgangsindividuums l_a liegen: $c \in [1, l_{\max} - l_a]$. Bei der reduzierenden Mutation darf analog dazu die Minimallänge des mutierten Individuums nicht unterschritten werden: $c \in [1, l_a - l_{\min}]$.

2.4 ZUSAMMENFASSUNG

Dieses Kapitel gab einen Überblick über das Gebiet der automatischen Testdatengenerierung für den Modelltest.

Zu Beginn wurden in Abschnitt 2.1 die Grundlagen für den Modell-Strukturtest beschrieben. Simulationsmodelle wurden zusammen mit ihrer Ausprägung als für diese Arbeit relevante SL/SF Modelle einführend beschrieben. Nach einer kurzen Einführung der Begriffe des Softwaretests im Allgemeinen und des Modelltests im Speziellen wurde auf strukturorientierte Testdatenermittlungsverfahren eingegangen. Diese basieren auf Fehlermodellen, welche auf der internen Struktur des Testobjekts beruhen: So wird von jenen internen Strukturelementen, wie beispielsweise bestimmten Anweisungen, Transitionen oder Zuständen erwartet, Fehler aufzufinden, wenn diese ausgeführt bzw. ausgelöst oder aktiviert werden. Die Qualität der strukturorientierten Tests wird anhand der erreichten strukturellen Überdeckung gemäß eines entsprechenden Überdeckungskriteriums gemessen.

*Modell-
Strukturtest*

Zur Automatisierung strukturorientierter Tests existiert eine Vielzahl von Verfahren, welche in Abschnitt 2.2 vorgestellt wurden. Diese lassen sich grob in statische und dynamische Verfahren unterteilen. Die in Abschnitt 2.2.1 vorgestellten statischen Verfahren basieren überwiegend auf symbolischer Ausführung und Constraint Solving, während die in Abschnitt 2.2.2 vorgestellten dynamischen Verfahren die Aufgabe der Testdatengenerierung in ein Optimierungsproblem transformieren, welches durch Anwendung eines robusten Suchverfahrens gelöst wird. Dazu kommen überwiegend evolutionäre Algorithmen zum Einsatz, welche aus diesem Grund gesondert in Abschnitt 2.3 beschrieben wurden.

Testautomatisierung

*Herausfor-
derungen*

Die Grenzen und Einschränkungen der existierenden Verfahren wurden in Abschnitt 2.2.4 analysiert und aufgezeigt. Es wurden drei grundlegende Probleme aufgezeigt, mit denen sich diese Arbeit in den beiden folgenden Kapiteln befassen wird.

Teil II

SUCHBASIERTER STRUKTURTEST VON SIMULINK MODELLEN

3

SIGNALGENERIERUNG UND -OPTIMIERUNG

*Die Lösung ist immer einfach,
man muss sie nur finden.*

— Alexander Solschenizyn

Dieses Kapitel stellt unterschiedliche Herangehensweisen zur Generierung und Optimierung von Signalen vor. Die vorgestellten Verfahren wurden entwickelt, um den dynamischen suchbasierten Strukturtest auch auf zustandsbehaftete Modelle anwenden zu können. Die in Abschnitt 2.2.4 auf Seite 44 herausgestellten Einschränkungen sollen somit adressiert werden.

Abschnitt 3.1 beschreibt zunächst drei unterschiedliche Signalgenerierungsansätze und vergleicht deren Effektivität anhand einer empirischen Fallstudie. Abschnitt 3.2 geht auf die beim Umgang mit Signalen notwendige spezielle Berechnung des Zielfunktionswerts ein. Ein komplexes Verfahren zur Berücksichtigung von Constraints im Signalgenerierungs- und -optimierungsprozess wird in Abschnitt 3.3 vorgestellt. Das Kapitel wird in Abschnitt 3.4 zusammengefasst.

Gliederung

3.1 ANSÄTZE

Die suchbasierte Generierung effektiver Testdaten in Form von Signalen für dynamische, zustandsbasierte Systeme ist im Allgemeinen und insbesondere für komplexe Systeme schwer zu realisieren. Dies liegt vor allem an ihren Anforderung an die Signale, eine Mindestlänge zu besitzen, also aus einer minimalen Anzahl von Zeitschritten zu bestehen. In Verbindung mit einer kleinen Abtastrate führt dies dazu, dass heutige Optimierungsverfahren aufgrund der resultierenden Suchraumgröße, d.h. der Anzahl der zu optimierenden Parameter, nicht anwendbar sind.

Motivation

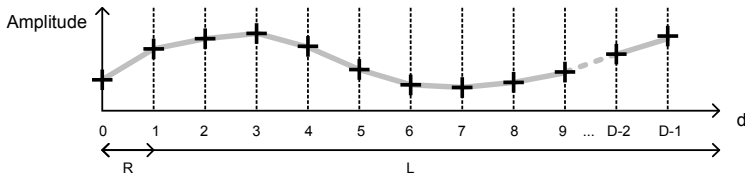


Abb. 19: Beispiel eines zeitdiskreten Signals mit der Abtastrate $R \in \mathbb{R}$ und der Länge $L \in \mathbb{R}$, bestehend aus $D \in \mathbb{N}_{>0}$ Datenpunkten.

Jedes Signal s , das für eine der $n \in \mathbb{N}_{>0}$ Systemeingaben optimiert werden soll, besteht aus insgesamt $\mathbb{N}_{>0} \ni D = L \cdot R^{-1}$ Datenwerten $v \in \mathbb{R}$, wobei $L \in \mathbb{R}$ die Signallänge und $R \in \mathbb{R}$ die Abtastrate beschreibt. Abb. 19 illustriert den Aufbau der Signale beispielhaft. Gilt es beispielsweise Testdaten für ein sehr kleines System mit $n = 3$ Eingaben, der gewünschten Signallänge von $L = 5\text{s}$ und einer Abtastrate von $R = 1000^{-1}\text{s}$ (1kHz) zu optimieren, ergeben sich bereits $n \cdot L \cdot R^{-1} = 15.000$ zu optimierende Datenwerte. Erfahrungsgemäß nimmt die Leistung heutiger metaheuristischer Optimierungsverfahren bereits ab 100 zu optimierenden Parametern deutlich ab.

Allgemeiner Ansatz

Es muss aus diesem Grund unter Verwendung eines geeigneten Dekoders – in diesem Fall eines Signalgenerators – wie in Abb. 20 illustriert, eine klare Unterscheidung zwischen dem geno- und dem phänotypischen Suchraum vorgenommen werden. Das Suchverfahren verwendet und optimiert eine einfache Datenstruktur im genotypischen Suchraum, welche mit Hilfe des Signalgenerators, unter Beachtung der verwendeten Repräsentation, in Lösungsvorschläge für das eigentliche Optimierungsproblem im phänotypischen Suchraum transformiert wird. Die Repräsentation beschreibt die Kodierung phänotypischer in genotypische Lösungsvorschläge (vgl. Abschnitt 2.3.1 auf Seite 49). Durch dieses Vorgehen, kann das Problem der Signalgenerierung und -optimierung von robusten Suchverfahren, wie z.B. von evolutionären Algorithmen, gelöst werden, da je nach gewähltem Dekoder eine starke Reduktion des genotypischen Suchraums stattfindet.

Gliederung

Die folgenden Kapitel stellen drei auf diesem Prinzip basierende Ansätze vor. Der erste Ansatz verwendet ein trigonometrisches Polynom, welches für seine Fähigkeit bekannt ist, beliebige mathematische Funktionen annähern zu können. Es wird beschrieben, wie

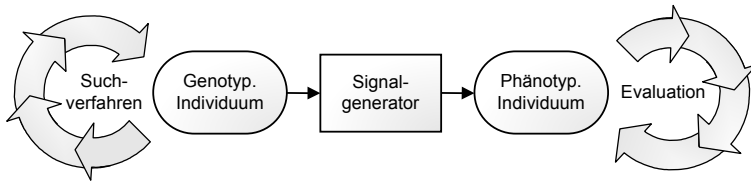


Abb. 20: Das verwendete Suchverfahren optimiert eine Datenstruktur im genotypischen Suchraum, welche mit Hilfe des Signalgenerators zur Evaluation in eine Lösung des Optimierungsproblems im phänotypischen Suchraum transformiert wird.

dieses unter Anwendung eines Optimierungsverfahrens zur Generierung und Optimierung von Signalen verwendet werden kann. Der zweite Ansatz verwendet eine Sequenz parametrierbarer Signalsegmente. Diese Parameter spezifizieren die Charakteristika der Signalsegmente und werden von genetischen Algorithmen optimiert. Auf diesem Ansatz aufbauend, wird eine Erweiterung vorgestellt, welche die Prinzipien und Operatoren der linearen genetischen Programmierung verwendet.

3.1.1 TRIGONOMETRISCHE POLYNOME

Dieser Ansatz verwendet zur Generierung von Signalen Polynome variablen Grades, insbesondere trigonometrische Polynome. Im Folgenden wird daher zunächst auf Polynome und anschließend auf trigonometrische Polynome und deren Verwendung für die Signalgenerierung eingegangen.

Die Funktion eines algebraischen Polynoms $f_k^P(x)$ mit dem Grad $k \in \mathbb{N}_{>0}$ ist eine parametrisierte Summe von $k + 1$ Potenzen der Variable x :

*Algebraische
Polynome*

$$f_k^P(x) = \sum_{i=0}^k a_i x^i. \quad (3.1)$$

Die Koeffizienten $a_i \in \mathbb{R}$ parametrieren das Polynom. Die Anzahl der zu spezifizierenden Koeffizienten hängt direkt vom Polynomgrad k ab: Sie ist gegeben durch $k + 1$.

Der Approximationssatz von Weierstraß [147] besagt, dass sich jede stetige Funktion f innerhalb eines abgeschlossenen und beschränkten Intervalls $[a, b]$ von einem algebraischen Polynom f_k^P

mit hinreichend großem Grad k beliebig gut approximieren lässt:

$$\left| f(x) - f_k^P(x) \right| < \epsilon \quad (3.2)$$

für $a \leq x \leq b$ und ein beliebig kleines $\epsilon \in \mathbb{R}$. Dies bedeutet im Umkehrschluss ebenfalls, dass sich durch geeignete Parametrierung des Polynoms jede beliebige stetige Funktion erzeugen lässt. Wertet man diese Funktion über ein gegebenes Zeitintervall aus, so erhält man ein Signal. Die Parametrierung eines Polynoms ermöglicht daher die Generierung von Signalen. Nach der Festlegung eines bestimmten Polynomgrads $k > 0$, können durch die Variation der sich ergebenden $k + 1$ Polynomkoeffizienten unterschiedliche Signale erzeugt werden.

Hoher Polynomgrad

Zur Erzeugung realitätsnaher Signalverläufe ist jedoch ein hoher Polynomgrad notwendig. Praktisch behindert demnach die damit einhergehende hohe Anzahl an zu optimierenden Polynomkoeffizienten eine erfolgreiche Anwendung algebraischer Polynome für die suchbasierte Generierung von Signalen. Algebraische Polynome sind darüber hinaus nicht dazu geeignet, periodische Funktionsverläufe zu approximieren. Eine Funktion $f : \mathbb{R} \rightarrow \mathbb{R}$ heißt periodisch mit der Periode $T \in \mathbb{R}_{>0}$, falls für alle $t \in \mathbb{R}$ gilt: $f(t + T) = f(t)$. Zur Approximation beliebiger periodischer Funktionen lassen sich trigonometrische Polynome verwenden: Weierstraß [147] wies nach, dass trigonometrische Polynome bei hinreichend großem Polynomgrad eine gegebene stetige 2π -periodische Funktion beliebig genau approximieren können.

Trigonometrische Polynome

Ein trigonometrisches Polynom ist eine endliche Linearkombination aus den trigonometrischen Funktionen Sinus und Kosinus. Gleichung 3.3 beschreibt die Berechnung der Funktionswerte $f_k^{TP}(t)$ eines trigonometrischen Polynoms für einen Zeitschritt t :

$$f_k^{TP}(t) = a_0 + \sum_{n=1}^k (a_n \cdot \cos(n \cdot t) + b_n \cdot \sin(n \cdot t)). \quad (3.3)$$

Das Polynom ist mittels der Koeffizienten $a_n, b_n \in \mathbb{R}$ parametrierbar. Der Parameter $k \in \mathbb{N}_{>0}$ definiert den Grad des Polynoms. Ein trigonometrisches Polynom ist eine periodische Funktion. Im Speziellen handelt es sich bei einem trigonometrischen Polynom um eine 2π -periodische Funktion, d.h. die Funktionswerte wiederholen sich alle 2π Zeitschritte. Die Periodizität des Polynoms lässt

sich durch Einführung des Parameters $\omega = \frac{2\pi}{T}$ auf eine beliebige reellwertige Zahl $T \in \mathbb{R}_{>0}$ festlegen. Die Gleichung 3.4 beschreibt die Berechnung der Datenwerte des trigonometrischen Polynoms:

$$f_k^{TP}(t) = a_0 + \sum_{n=1}^k (a_n \cdot \cos(n \cdot \omega \cdot t) + b_n \cdot \sin(n \cdot \omega \cdot t)). \quad (3.4)$$

Jede stetige und periodische Funktion kann – bei genügend großem Polynomgrad k – durch diese endliche Linearkombination aus Sinus- und Kosinusfunktionen approximiert werden. Analog zu algebraischen Polynomen, kann das Festlegen eines Polynomgrades und die Variation der insgesamt $2 \cdot k + 1$ resultierenden Polynomkoeffizienten zur Generierung von Signalen verwendet werden.

SUCHBASIERTE SIGNALGENERIERUNG

In Abschnitt 3.1 auf Seite 67 wurde die zugrundeliegende Unterscheidung zwischen geno- und phänotypischen Individuen beschrieben. Im Falle der trigonometrischen Polynome besteht ein genotypisches Individuum aus den insgesamt $2k + 1$ Koeffizienten des Polynoms – bei zuvor definiertem Polynomgrad k . Diese Parameter werden vom angewandten Suchverfahren variiert. Ein phänotypisches Individuum, d.h. ein konkretes Testdatum, mit dem das Testobjekt ausgeführt wird, ist durch das Signal gegeben, welches durch Gleichung 3.4 unter Verwendung des genotypischen Individuums berechnet wird.

*Geno- &
phänotyp.
Individuen*

Abhängig von der Natur der zu generierenden Signale muss der Skalierungskoeffizient der Polynomperiodizität ω angepasst werden. Sind keine Wiederholungen erwünscht, so kann für den enthaltenen Parameter T die Länge des zu generierenden Signals angegeben werden. Aufgrund der Periodizität des trigonometrischen Polynoms beginnt und endet das Signal beim selben Amplitudenwert. Ist dies nicht erwünscht, muss die Periode des Polynoms auf einen größeren Wert gesetzt werden: 20 – 40% der gewünschten Signallänge sind eine erste Orientierung. Das Signal wird dann jedoch nur für die gewünschten Datenpunkte ausgewertet. Dieses Vorgehen ist in Abb. 21 beispielhaft illustriert. Ein trigonometrisches Polynom mit dem Grad 8 wurde jeweils verwendet, um eine Stufenfunktion zu generieren (grau hinterlegt); Die gewünschte Signallänge beträgt 100ms bei einer Abtastrate

Periodizität

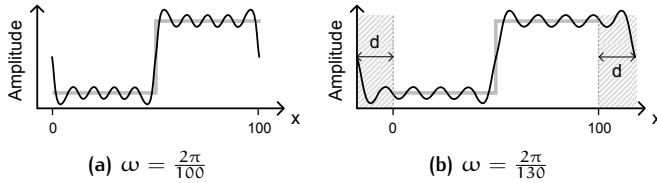


Abb. 21: Variierende Skalierung der Periodizität eines trigonometrischen Polynoms zur Vermeidung gleicher Amplitudenwerte am Anfang und am Ende des generierten Signals.

von 1kHz. In Abb. 21a ist das Ergebnis mit einem Skalierungskoeffizienten $\omega = \frac{2\pi}{100}$ dargestellt. Es ist zu sehen, dass das erzeugte Signal am Anfang wie auch am Ende den selben Amplitudenwert besitzt. Demgegenüber gilt für das Polynom in Abb. 21b: $\omega = \frac{2\pi}{130}$. Das Signal wurde um 30% ($2 \cdot d$) gestreckt und entsprechend um 15% (d) nach links verschoben. Durch ausschließliche Auswertung im gewünschten Bereich, d.h. durch Abschneiden der irrelevanten Bereiche, ergibt sich das gewünschte Signal. Hervorzuheben ist an dieser Stelle das für trigonometrische Polynome charakteristische Über- und Unterschwingen bei linearen Signalverläufen.

3.1.2 SEGMENTBASIERTER ANSATZ

Idee Die zugrundeliegende Idee des segmentbasierten Ansatzes entstammt den Arbeiten von Baresel *et al.* und Griep [13, 53]. Diese beschreiben einen einfachen Weg, Signale durch Aneinanderreihung einer bestimmten Anzahl parametrierter Signalsegmente zu generieren. Im Folgenden bezeichnet Σ die Menge aller Signalsegmente σ , aus denen ein Signal zusammen gesetzt ist ($\sigma \in \Sigma$). Als Signalsegmente kommen spezifische Basissignale zum Einsatz. Abb. 22 auf der nächsten Seite illustriert die Aneinanderreihung parametrierter Basissignale (Signalsegmente) zu einem zusammengesetzten Signal. Jedes dieser Signalsegmente kann mit Hilfe der drei Parameter *Amplitude*, *Transitionstyp* und *Breite* parametrierter werden.

Amplitude Der Amplitudenwert $a : \Sigma \rightarrow \mathbb{R}$ eines jeden Signalsegments ist beschränkt auf die für die entsprechende Eingabe des Testobjekts (und somit für das jeweilige Signal) spezifizierten Amplituden-

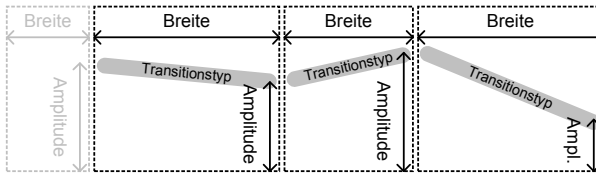


Abb. 22: Allgemeiner Signalaufbau basierend auf einer bestimmten Anzahl parametrierter Basissignale. Jedes dieser Basissignale beruht auf drei Parametern: Amplitude, Transitionstyp und Breite.

beschränkungen. Im Speziellen beruht jedes Signalsegment auf zwei Amplitudenwerten: einem Start- und einem Endamplitudenwert. Der Startamplitudenwert entstammt dem Amplitudenwert des vorherigen Signalsegments. Der Endamplitudenwert hingegen hängt von der Parametrierung des Signalsegments selbst ab. Aus diesem Grund stellt das erste Signalsegment lediglich den Startamplitudenwert für das darauf folgende Signalsegment zur Verfügung. Es wird darüber hinaus nicht zur Signalgenerierung verwendet.

Der durch einen Identifikator (T) spezifizierte Transitionstyp $tt : \Sigma \rightarrow T$ der Signalsegmente bestimmt die Art, mit der die Startamplitude mit der Endamplitude verbunden ist und hat somit einen starken Einfluss auf die Natur des generierten Signals. Zur Verfügung stehen fünf Transitionstypen, auf die im Folgenden genauer eingegangen wird. Dieser Ansatz ist – falls nötig – mit weiteren Transitionstypen erweiterbar.

Transitions-
typ

Die Parametrierung der Breite $b : \Sigma \rightarrow \mathbb{R}$ eines Signalsegments sichert die Variabilität der generierten Signale. Falls eine einzuhaltende Signallänge spezifiziert wurde, werden die Breiten aller involvierten Signalsegmente derart skaliert, dass sie in ihrer Gesamtheit dieser spezifizierten Signallänge entsprechen. Anderenfalls können die Breiten auch ohne Skalierung verwendet werden, was zu einer Variabilität der generierten Signallängen führt.

Breite

Zur Signalgenerierung sind also grundlegende Information über die Natur der zu generierenden Signale notwendig, welche vom Tester zur Verfügung gestellt werden müssen. Neben den generellen Attributen, wie beispielsweise die Länge der zu generierenden Signale und ihre gewünschte Signalaufösung, kann dies einzelne Spezifikationen der Signale für die einzelnen Eingaben des Testobjekts beinhalten. So kann für jedes Signal eine Liste von

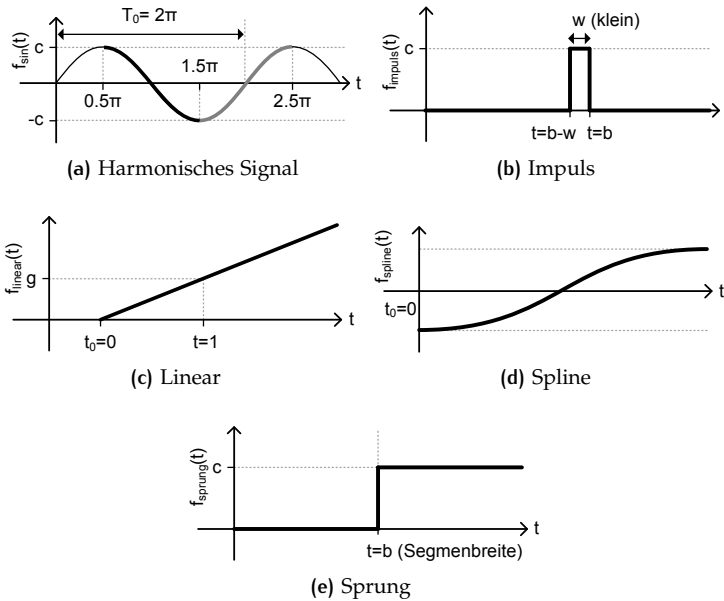


Abb. 23: Fünf Basissignale, welche im segmentbasierten Ansatz zur Erzeugung von Signalen verwendet werden.

Basissignalen spezifiziert werden, aus denen es zusammengesetzt sein soll. Ebenso können die einzuhaltenden Amplitudenbeschränkungen angegeben werden.

BASISSIGNALE

Dieser Abschnitt beschreibt eine Auswahl von fünf Basissignalen, welche für die Regelungs- und Systemtechnik von besonderer Relevanz sind [53].

Jedes dieser in Abb. 23 illustrierten elementaren Signale wird vom segmentbasierten Ansatz zum Aufbau von Signalen verwendet.

SINUS Dieses Signalsegment ($T = \text{Sinus}$) basiert auf der Sinusfunktion, deren erste Ableitung (Steigung) sowohl zu Beginn als auch am Ende des Segments Null ist, um die Monotonie

des generierten Signals zu gewährleisten. Dieses Signalsegment besteht daher aus einem Viertel der Sinuskurve.

$$f_{\sin}(t) = c \cdot \sin\left(s \cdot t + \frac{\pi}{2}\right) + c + a_S. \quad (3.5)$$

Der Koeffizient $c \in \mathbb{R}$ beeinflusst die Steigung der Sinusfunktion und ist definiert als $c = \frac{a_E - a_S}{2}$. Der Faktor $s \in \mathbb{R}$ streckt bzw. staucht die Sinuskurve entsprechend der gewünschten Breite b des Signalsegments und ist definiert als $s = \frac{\pi}{b/R}$. Die Parameter a_S und a_E spezifizieren jeweils die Anfangs- und Endamplitude des Signalsegments.

Das Sinus-Signalsegment kann immer dann verwendet werden, wenn ein harmonischer Übergang zwischen zwei Amplitudenwerten realisiert werden soll. Die Erzeugung eines Geschwindigkeitssignals eines Kraftfahrzeugs ist ein Beispiel für dessen Anwendung.

SPLINE Die Spline-Funktion ($T = \text{Spline}$) ist ein Polynom dritten Grades, welches zum Beginn und zum Ende des Signalsegments die Steigung Null besitzt und jeweils die gewünschten Amplitudenwerte aufweist. Es lässt sich bestimmen durch Lösung des folgenden Gleichungssystems:

$$\begin{aligned} a_S &= a \cdot x_1^3 + b \cdot x_1^2 + c \cdot x_1 + d \stackrel{x_1=0}{=} d \\ a_E &= a \cdot x_2^3 + b \cdot x_2^2 + c \cdot x_2 + d \\ 0 &= 3 \cdot a \cdot x_1^2 + 2 \cdot b \cdot x_1 + c \stackrel{x_1=0}{=} c \\ 0 &= 3 \cdot a \cdot x_2^2 + 2 \cdot b \cdot x_2 + c \end{aligned} \quad (3.6)$$

Die Parameter a , b und c sind die Polynomkoeffizienten, die es durch Lösen des Gleichungssystems zu bestimmen gilt. x_1 ist der Start- und x_2 der Enddatenpunkt des Signalsegments. Da alle Signalsegmente beim Datenpunkt 0 beginnen, gilt $x_1 = 0$.

Das Spline Signal erlaubt wie auch das harmonische Signal einen flüssigen Übergang zwischen zwei Amplitudenwerten. Die Anwendungsmöglichkeiten für das Spline Signal entsprechen denen des harmonischen Signals. Aufgrund ihrer Ähnlichkeit kann das Spline Signal und das harmonische Signal alternativ verwendet werden.

LINEAR Das lineare Signalsegment ($T = \text{Linear}$) ist sehr simpel und zur Darstellung auf- und absteigender Signalverläufe von besonderer Relevanz.

$$f_{\text{linear}}(t) = g \cdot t. \quad (3.7)$$

Der verwendete Steigungsparameter kann gemäß $g = \frac{a_E - a_S}{dp}$ berechnet werden. Der Divisor dp enthält die Anzahl der Datenpunkte des Signalsegments.

Verwendung findet das lineare Signalsegment beispielsweise bei der Erzeugung eines Signals, welches die sich regelmäßig verändernde Position eines Beschleunigungs- oder Bremspedals eines Kraftfahrzeugs darstellt.

SPRUNG Das Sprung-Signalsegment ($T = \text{Sprung}$) ist von besonderer Wichtigkeit für die Erzeugung unstetiger Signalverläufe, wie sie in digitalen Systemen häufig vorkommen.

$$f_{\text{sprung}}(t) = \begin{cases} a_S & , \text{ falls } t < b \\ a_E & , \text{ falls } t = b. \end{cases} \quad (3.8)$$

Das Signalsegment besitzt bis auf den letzten Datenpunkt b den Datenwert der Startamplitude a_S . Im letzten Datenpunkt (b) besitzt es schließlich den Datenwert der Endamplitude a_E . Werden mehrere Sprung-Signalsegmente hintereinander gereiht, ergibt sich ein unstetiger Signalverlauf.

Dieses sprungförmige Signalsegment kann verwendet werden, wann immer sprungförmige Amplitudenänderungen des Signals auftreten. Beispielsweise zur Erzeugung schwellwertbasierter Signale.

IMPULS Das impulsförmige Signalsegment ($T = \text{Impuls}$) realisiert einen kurzzeitigen Amplitudenausschlag innerhalb des erzeugten Signals. Es ist wie folgt definiert:

$$f_{\text{impuls}}(t) = \begin{cases} a_S & , \text{ falls } t < b - w \\ a_E & , \text{ falls } t \geq b - w. \end{cases} \quad (3.9)$$

Der Parameter w spezifiziert die Länge des zu erzeugenden Impulses – gegeben als Anzahl der Datenpunkte. Ähnlich dem sprungförmigen Signalsegment wird der Impuls am

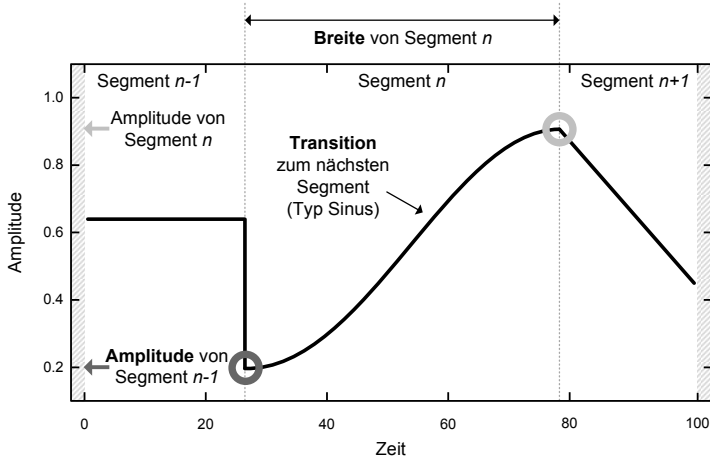


Abb. 24: Illustration der Verknüpfung eines beispielhaften mit *Amplitude*, *Breite* und *Transitionstyp* parametrisierten Signalsegments mit seinen umgebenden Signalsegmenten. Das Signalsegment n verläuft über die Breite von $b(n) = 50\text{s}$ mit Transitionstyp $tt(n) = \text{Sinus}$ von Amplitudenwert $a(n-1) = 0.2$ zu $a(n) = 0.92$.

Ende des Segments erzeugt. Der einzige Unterschied besteht darin, dass das nachfolgende Signalsegment mit der Startamplitude dieses Impulssegments beginnt und nicht wie in allen anderen Fällen mit dessen Endamplitude.

Das impulsförmige Signalsegment wird immer dann verwendet, wenn kurzzeitige Amplitudenausschläge erzeugt werden müssen. Das Drücken eines Druckknopfes oder das Betätigen eines Hebels sind beispielsweise Vorgänge, für welche Impulse sinnvoll sind.

SIGNALKONSTRUKTION

Der segmentbasierte Signalgenerator erzeugt Signale durch die Aneinanderreihung einer bestimmten Anzahl von Signalsegmenten. Jedes dieser Signalsegmente parametrisiert eines der Basissignale mit den drei Parametern *Amplitude*, *Breite* und *Transitionstyp*. Sind diese Parameter für alle Signalsegmente bekannt, können die Signalsegmente gerendert und miteinander zu einem gesamten Signal verknüpft werden.

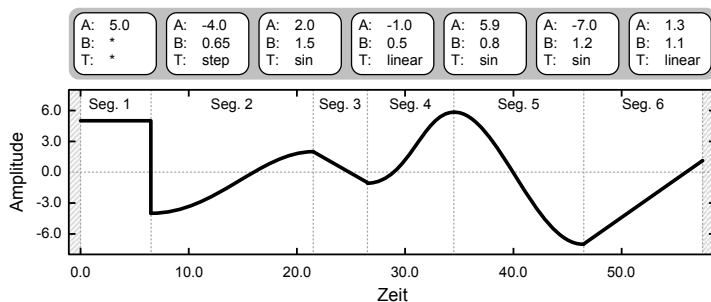


Abb. 25: Konstruktion eines Signals durch Aneinanderreihung einzelner Signalsegmente. Oben ist ein beispielhaftes Chromosom mit sieben Genen dargestellt, während das daraus erzeugte Signal darunter dargestellt ist.

*Verknüpfung von
Signalsegmenten*

Diese Verknüpfung ist in Abb. 24 beispielhaft für ein Signalsegment n und dessen angrenzende Signalsegmente illustriert. Die Breite des Signalsegments n wird direkt vom entsprechenden Parameter $b(n)$ bestimmt. Wie zuvor beschrieben, beginnt der Signalverlauf des betrachteten Signals im Amplitudenwert des vorherigen Signalsegments. Die Startamplitude a_S für Signalsegment n lautet demnach $a(n-1)$. Demgegenüber ergibt sich die Endamplitude a_E des Signalsegments aus dessen Parametrierung: $a(n)$. Die Art, wie das Signal von a_S nach a_E verläuft, hängt wiederum vom gewählten Transitionstyp ab. Im angegebenen Beispiel wird für Segment n ein sinusförmiger Amplitudenübergang verwendet.

Beispiel

Die Signalkonstruktion für ein beispielhaftes Chromosom, d.h. für eine Lösung im genotypischen Suchraum, bestehend aus sieben Signalsegmenten, ist in Abb. 25 illustriert. Das resultierende Signal besteht aus sechs gerenderten Signalsegmenten. Es sollte ein Signal mit einer Gesamtlänge von 58 Sekunden erzeugt werden. Das erste gerenderte Signalsegment verläuft schrittweise von Amplitudenwert 5 zu Amplitudenwert -4 . Von diesem Punkt aus beginnt das zweite Signalsegment und verläuft mit einem harmonischen Übergang zu Amplitudenwert 2 und so weiter. Der Einfluss des Parameters *Breite* der Signalsegmente kann deutlich am erzeugten Signal abgelesen werden. So führt beispielsweise die dreifache Breite des zweiten Signalsegments in Relation zum dritten Signalsegment im genotypischen Suchraum zu ei-

ner Verdreifachung der Breite des entsprechenden gerenderten Signalsegments im phänotypischen Suchraum.

SIGNALOPTIMIERUNG

Wie zuvor beschrieben, kann ein Signal konstruiert werden, basierend auf den Parametern der Signalsegmente, aus denen es zusammengesetzt ist. Es handelt sich um genau $3 \cdot n$ Parameter, wobei n die Anzahl der enthaltenen Signalsegmente beschreibt. Die Anzahl der Parameter kann jedoch reduziert werden, falls der Benutzer in der Lage ist, Informationen über die zu generierenden Signale zur Verfügung zu stellen. Falls beispielsweise nur ein einziger Transitionstyp für ein Signal erlaubt ist, reduziert sich die Anzahl der Parameter um n . Es würden in diesem Fall $2 \cdot n$ Parameter zur Beschreibung eines Signals genügen. Das selbe gilt für die Breite: Diese kann auf den selben Maximal- und Minimalwert beschränkt und somit fixiert werden, wodurch die Anzahl der Parameter um n reduziert werden würde. Der genotypische Suchraum ist durch diese Parameter definiert.

*Genotyp.
Suchraum*

Demgegenüber ergeben sich die Lösungsvorschläge im phänotypischen Suchraum (die zur Ausführung des Testobjekts verwendeten Signale), durch die Signalgenerierung unter Verwendung der genotypischen Lösungsvorschläge (der zuvor beschriebenen Parameter).

*Phänotyp.
Suchraum*

3.1.3 LÄNGENVARIABLER SEGMENTBASIERTER ANSATZ

Der zuvor vorgestellte segmentbasierte Ansatz hat den Nachteil, dass die Anzahl der verwendeten Signalsegmente je Signal vor dem Suchlauf festgelegt werden muss. Der längenvARIABLE segmentbasierte (LvSb) Ansatz erweitert den segmentbasierten Ansatz durch Anwendung grundlegender Operatoren der linearen genetischen Programmierung. Dadurch kann die Anzahl der je Signal verwendeten Signalsegmente automatisch variiert werden. Es ergibt sich dadurch eine höhere Variabilität der Signaldynamik und die Signallänge kann ebenfalls dynamisch variiert werden. Die Notwendigkeit der manuellen Festlegung der Segmentanzahl wird obsolet.

Motivation

Die lineare genetische Programmierung wird üblicherweise zur

LGP

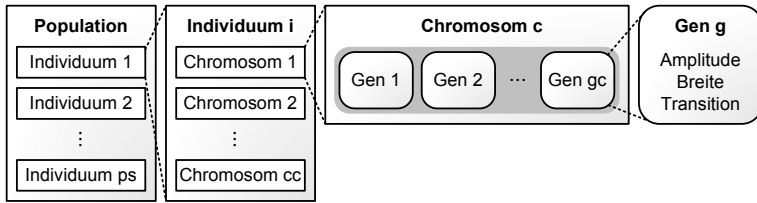


Abb. 26: Kodierung eines genotypischen Individuums für den längenvariablen segmentbasierten Ansatz. Eine Population besteht aus ps Individuen, jedes Individuum besteht aus cc Chromosomen, welche wiederum aus gc Genen bestehen.

Generierung linear kodierter Computerprogramme verwendet (vgl. Abschnitt 2.3.2 auf Seite 57). Bei der Aneinanderreihung von Signalsegmenten handelt es sich ebenfalls um lineare Datenstrukturen. Daher können die grundlegenden Prinzipien und Operatoren der linearen genetischen Programmierung zur Optimierung von segmentbasierten Signalkodierungen verwendet werden.

*Kodierung
der Daten-
struktur*

In dieser Arbeit wurde eine spezielle Datenstruktur entwickelt, die in Abb. 26 illustriert ist. Der Algorithmus arbeitet, wie für evolutionäre Algorithmen typisch, auf einer Population P von Individuen i (vgl. Abschnitt 2.3.1 auf Seite 49). Jedes Individuum repräsentiert einen Lösungsvorschlag für das jeweilige Überdeckungsziel. Jedes dieser Individuen besteht aus einer bestimmten Anzahl von Chromosomen. Ein Individuum besitzt für jede Eingabe des Testobjekts genau ein Chromosom. Ein Chromosom ist eine spezielle Repräsentation eines gültigen Signals für eine Eingabe des Testobjekts. Ein Chromosom besteht aus einer bestimmten Anzahl von Genen. Jedes Gen entspricht genau einem Signalsegment und enthält daher die drei Attribute *Amplitude*, *Breite* und *Transition*.

Alle signalspezifischen Attribute, wie z.B. Amplitudenschranken oder gültige Transitionstypen, gelten jeweils für das gesamte Chromosom und finden somit Anwendung bei allen darin enthaltenen Genen. Die Chromosome repräsentieren die lineare Datenstruktur, auf der die Operatoren der linearen genetischen Programmierung operieren. Diese Datenstruktur wird gemäß der für die L-GP typischen Varianten der Operatoren Selektion, Rekombination und Wiedereinfügen optimiert (vgl. Abschnitt 2.3.2 auf Seite 57).

SIGNALOPTIMIERUNG

Der genotypische Suchraum ist beim längenvariablen segmentbasierten Ansatz definiert durch die in Abb. 26 illustrierte Datenstruktur.

*Genotyp.
Suchraum*

Die Lösungen im phänotypischen Suchraum werden vom zugrundeliegenden Signalgenerator generiert. Zur Erstellung von Signalen werden die Lösungen aus dem genotypischen Suchraum wie beim segmentbasierten Ansatz interpretiert.

*Phänotyp.
Suchraum*

3.1.4 EMPIRISCHER VERGLEICH

Dieser Abschnitt beschreibt eine kleine experimentelle Fallstudie zur Beurteilung der Fähigkeit der drei vorgestellten Signalgenerierungsverfahren, Signale zu generieren.

Im Rahmen des evolutionären Tests sucht das verwendete Suchverfahren nach einem spezifischen Satz von Signalen, welcher als Teststimuli für das Testobjekt verwendet wird. Diese Suche wird durch die angewandte Zielfunktion gesteuert. Durch Bewertung der generierten Teststimuli wird der Signalgenerierungs- und Optimierungsprozess von der Zielfunktion direkt beeinflusst. Die Zielfunktion spezifiziert indirekt genau jenen Signalsatz, welcher zu einem gewünschten Verhalten des Testobjekts führt und berechnet die Distanz zum jeweils evaluierten Teststimulus. Aus diesem Grund besteht eine einfache und zeitsparende Methode zur Beurteilung der Fähigkeit eines Signalgenerierungsansatzes in der Anwendung für die Approximation eines spezifischen Signals.

Die zu diesem Zweck zu approximierenden Signale werden im Folgenden kurz beschrieben. Anschließend wird der verwendete experimentelle Aufbau beschrieben, bevor auf die Ergebnisse der Fallstudie eingegangen wird.

TESTOBJEKTE

Sechs realistische Signalverläufe aus dem Automobilbereich stehen für diese experimentelle Fallstudie zur Verfügung. Sie entstammen unterschiedlichen Abteilungen der Daimler AG und stellen eine repräsentative Auswahl realer Signalverläufe dar. Bei

*Signal-
auswahl*

der Auswahl der Signale wurde darauf geachtet, dass eine gewisse Diversität der Signaleigenschaften vorhanden ist. So sollten beispielsweise verrauschte Signale berücksichtigt werden, stufenförmige und stetige Signale vorhanden sein und Signale mit hoher bzw. niedriger Dynamik Berücksichtigung finden.

*Eigen-
schaften*

Jedes dieser Signale wurde direkt vom CAN-Bus des Fahrzeugs aufgezeichnet und beschreibt entweder die Ausgabe eines bestimmten Sensors, wie z.B. ein Temperatursensor, oder intern berechnete Informationen, wie beispielsweise der innerhalb eines Abstandsregeltempomaten berechnete Abstand zu einem vorausfahrenden Fahrzeug. Alle Signale wurden für eine Zeitspanne von 30 Sekunden aufgezeichnet und besitzen eine Abtastrate von einer Millisekunde.

EXPERIMENTELLER AUFBAU

*Verglichene
Ansätze*

Für diese experimentelle Fallstudie wird die Signalgenerierung und -optimierung vergleichend mit dem Ansatz der trigonometrischen Polynome und mit dem längenvariablen segmentbasierten Ansatz durchgeführt (vgl. Abschnitte 3.1.1 und 3.1.3). Zusätzlich werden die Experimente zu Kontrollzwecken mit der Zufallssuche durchgeführt. Dadurch wird die Zielführung der angewandten Optimierungsverfahren erkennbar. Die Zufallssuche wird realisiert durch Verwendung des LvSb-Ansatzes und wiederholter Erstellung und Auswertung von initialen Populationen. Der segmentbasierte Ansatz wird hier nicht verwendet, da der einzige Unterschied zum LvSb-Ansatz darin besteht, dass die Anzahl der Signalsegmente im Voraus spezifiziert werden muss.

*Konfigu-
ration
Polynom*

Für den Ansatz mit trigonometrischen Polynomen kommt für die Suche im genotypischen Suchraum ein evolutionärer Algorithmus zum Einsatz. Dessen Konfiguration wurde der Arbeit von Windisch *et al.* entnommen [150]. Dieser Parametersatz wurde bereits erfolgreich für den Strukturtest eingesetzt. Es wurde ein Polynomgrad von 15 gewählt, da sich dieser in zuvor ausgeführten Experimenten als besonders vorteilhaft hinsichtlich der Signaldarstellung und der resultierenden Suchraumgröße erwiesen hat. Es ergibt sich entsprechend ein 31-dimensionaler Suchraum. Alle 31 Parameter werden gleichverteilt zufällig im Intervall $[-100, 100]$ initialisiert. Die Optimierung wird für insgesamt 200 Iterationen durchgeführt, weshalb bei der gewählten Populationsgröße von

Parameter		Wert
Initialisierung	Individuen	50
	Gene	$\in [60, 200]$
Selektion	Variante	SUS, Turnier ($p \in [0.1, 0.5]$)
Rekombination	Variante	Homolog, Zwei-Punkt
	P_R	0.75
	Generationslücke	1.0
Mutation	Variante	Reduzierend ($p = 0.1$), Mehrpunkt ($p = 0.8$), Erweiternd ($p = 0.1$)
	P_{M_i}	0.95
	P_{M_c}	0.9
	P_{M_g}	0.05
	Schritt	0.1
Wiedereinfügen	Variante	Elitär
	Rate	0.9
Abbruch	Generationen	200
	Fitness	≤ 0

Tab. 2: Konfiguration der genetischen Algorithmen des längenvariablen segmentbasierten Ansatzes für die experimentelle Fallstudie.

40 Individuen insgesamt 8.000 Zielfunktionsauswertungen durchgeführt werden.

Tab. 2 enthält die Konfiguration, die für die Experimente mit dem LvSb-Ansatz verwendet werden. Der Signalgenerator wurde auf die Verwendung von Sinus, Linear und Sprung als zulässige Transitionstypen konfiguriert und für die Amplitudenschranken wurde das Intervall $[-150, 150]$ verwendet.

*Konfiguration
LvSb*

Für jedes zu approximierende Signal sollen insgesamt je Ansatz zehn komplette Optimierungsläufe ausgeführt werden. Das Ergebnis aller Läufe wird gemittelt. Der Zielfunktionswert (die Fitness) eines Individuums wird berechnet als der mittlere quadratische Abstand des aus dem Individuum generierten Signals zum Zielsi-

Ablauf

gnal. Die Ergebnisse der Läufe, d.h. die Signale, die vom jeweils besten Individuum generiert wurden, werden zur Untersuchung und Auswertung gespeichert.

ERGEBNIS

Das Ergebnis des durchgeführten Experiments ist in Abb. 27 für jedes einzelne Signal dargestellt. In jeder der enthaltenen Grafiken ist das zu approximierende Zielsignal grau gezeichnet, während die vom jeweils besten Individuum der untersuchten Verfahren generierten Signale in Schwarz gezeichnet sind. Erwartungsgemäß lieferte die Zufallssuche die vergleichsweise schlechteste Approximationsleistung bei allen untersuchten Signalen.

- Signal 1* Signal 1 besitzt einen relativ hohen Rauschanteil und wie zu erwarten war, waren beide Verfahren nicht in der Lage, ein ähnlich verrauschtes Signal zu generieren. Theoretisch wäre eine beliebig gute Approximation unter Verwendung eines trigonometrischen Polynoms mit hinreichend großem Polynomgrad möglich. Das selbe gilt für den LvSb-Ansatz bei Verwendung von linearen Transitionstypen und einer hinreichend großen Anzahl von Signalsegmenten. Da der Polynomgrad auf 15 fixiert und die zulässige Anzahl der Signalsegmente auf 60 bis 200 festgelegt wurde, war von beiden Verfahren lediglich eine grobe Annäherung möglich.
- Signal 2* Signal 2 weist mit dem konstanten Funktionswert von -22.0 in 75% aller Datenpunkte einen relativ simplen Signalverlauf auf. Im ersten Drittel besitzt es zwei sinusförmige Signalabstiege und zum Zeitpunkt 25s einen kleinen impulsähnlichen Signalsprung. Beide Verfahren waren in der Lage das Signal gut anzunähern, einzige Ausnahme ist der erwähnte Signalsprung. Grund dafür ist, dass die Fitness mit Hilfe der Zielfunktion jeweils in Abhängigkeit des gesamten Signals berechnet wird. Diese kleinen und kurzfristigen impulsähnlichen Ausschläge haben daher einen nur geringen Einfluss. Darüber hinaus ist erkennbar, dass der Polynom-Ansatz nicht in der Lage war, ein vergleichbar konstantes Signal zu erzeugen, da dazu ein deutlich höherer Polynomgrad von Nöten gewesen wäre.
- Signal 3* Signal 3, größtenteils ähnlich zum vorherigen Signal, weist einen konstanten Signalverlauf auf. Der interessante Teil befindet sich innerhalb der ersten fünf Sekunden, in denen ein zweifaches schnelles Ansteigen und Abfallen des Signals beobachtet werden

kann. Beide Signalgenerierungsverfahren waren nicht in der Lage, diesen ersten herausfordernden Teil des Signals anzunähern.

Sowohl Signal 4 als auch Signal 6 besitzen eine sprungförmige Signalcharakteristik. Obwohl Signal 4 sehr simpel aussieht, ist die Approximationsqualität niedriger als erwartet. Der steile Signalanstieg innerhalb der ersten drei Sekunden beispielsweise wurde von beiden Verfahren nicht genügend gut angenähert. Die selben Probleme konnten bei den kurzen Signalabstiegen in der Mitte des Signals beobachtet werden. Für Signal 6 wurde aufgrund seiner Einfachheit von beiden Verfahren eine gute Approximation erreicht.

*Signale
4 und 6*

Die Schwierigkeit bei der Approximation von Signal 5 liegt in der Annäherung der beiden Täler zu Beginn des Signalverlaufs und die beiden dynamischen impulsähnlichen Signaländerungen am Ende. Während der LvSb-Ansatz in Lage war, die Täler gut zu approximieren, waren die beiden anderen Verfahren nicht in der Lage, sowohl die hochdynamischen Teile des Signals als auch den konstanten Signalverlauf in der Mitte anzunähern.

Signal 5

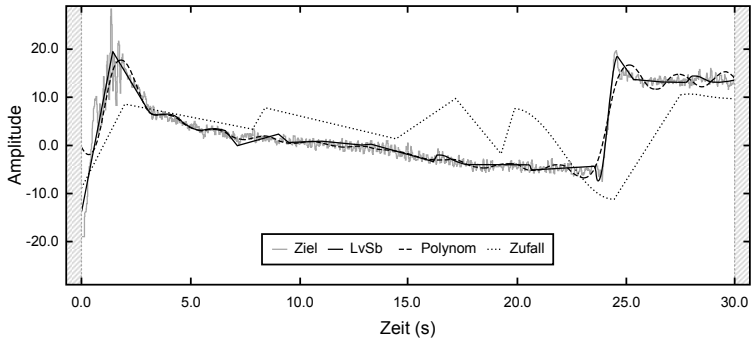
Zusätzlich vergleicht Abb. 28 die Effektivität der zugrundeliegenden Suche der entwickelten Signalgenerierungsverfahren mit der Zufallssuche. Dargestellt ist für jedes zu approximierende Signal der über alle Läufe gemittelte jeweils beste Zielfunktionswert. Sowohl der Polynom- als auch der LvSb-Ansatz übertreffen den Zufallstest erwartungsgemäß bei Weitem. Beide Verfahren erzielen vergleichbare Ergebnisse mit einem leichten Vorteil für den LvSb-Ansatz. In drei Fällen hat der Polynom-Ansatz einen besseren, d.h. kleineren, gemittelten Zielfunktionswert gefunden, während der LvSb-Ansatz in den verbleibenden drei Fällen überlegen war. Um die statistische Signifikanz der beobachteten Unterschiede zu beurteilen, wurde für jedes Experiment ein U-Test (Mann-Whitney-Wilcoxon) durchgeführt. Das Ergebnis dieses Tests verdeutlicht, dass ein Großteil der Effektivitätsunterschiede statistisch signifikant ist. Fünf der Unterschiede haben sich als statistisch signifikant herausgestellt. Lediglich der Unterschied bei Signal 4 war statistisch nicht signifikant.

*Vergleich
der
Effektivität*

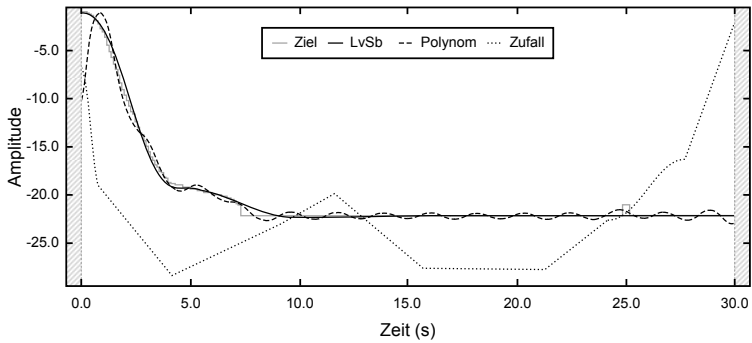
*Statistische
Signifikanz*

Die Experimente haben verdeutlicht, dass die vorgestellten suchbasierten Signalgenerierungsverfahren erfolgreich reale Signale annähern und somit erzeugen können, allerdings noch Schwierigkeiten haben, bestimmte Signalcharakteristika zu erzeugen. Der Polynom-Ansatz kann sehr erfolgreich zur Generierung einfacher

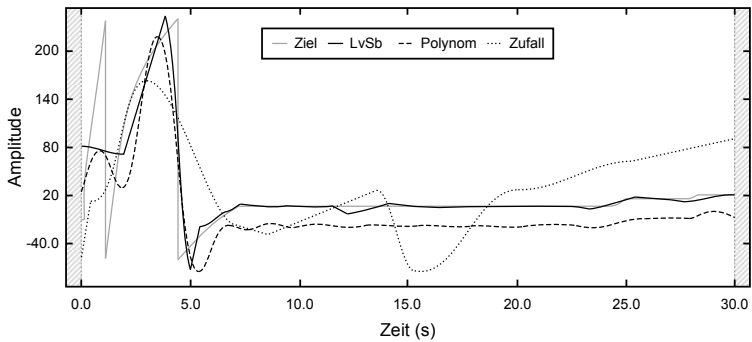
*Aus-
wertung*



(a) Signal 1

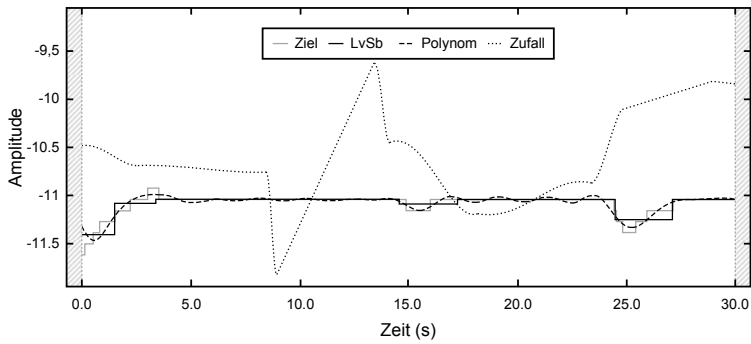


(b) Signal 2

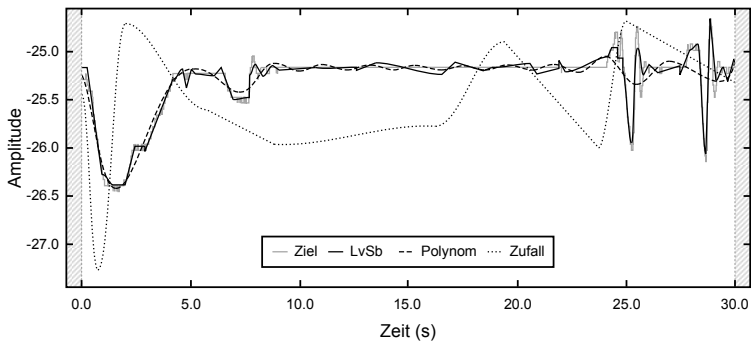


(c) Signal 3

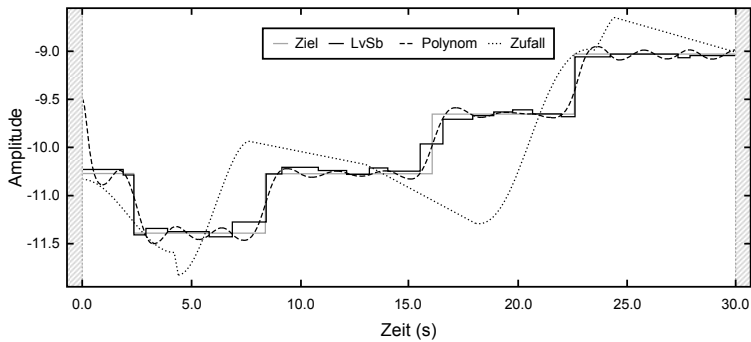
Abb. 27: Ergebnis der experimentellen Fallstudie zur vergleichenden Analyse der entwickelten Signalgenerierungsansätze durch Approximation von sechs realen Signalverläufen.



(d) Signal 4



(e) Signal 5



(f) Signal 6

Abb. 27 (Fs.): Ergebnis der experimentellen Fallstudie zur vergleichenden Analyse der entwickelten Signalgenerierungsansätze durch Approximation von sechs realen Signalverläufen.

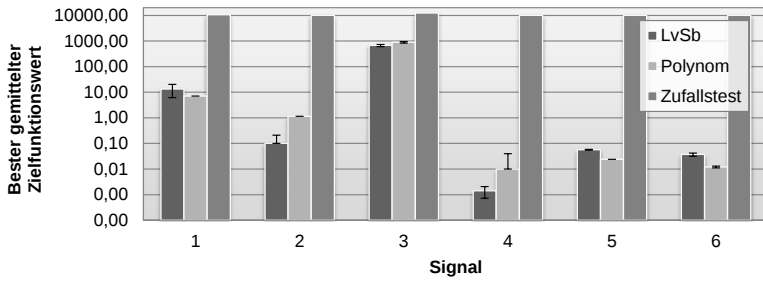


Abb. 28: Vergleich der Effektivität der untersuchten Signalgenerierungsverfahren für die durchgeführte Fallstudie.

Signalen verwendet werden. Besonders dynamische Signaleigenschaften können jedoch lediglich durch eine starke Anhebung des Polynomgrads erzeugt werden. Dies führt zu einer deutlichen Vergrößerung des genotypischen Suchraums und verringert dadurch die Wahrscheinlichkeit, eine gute Lösung zu finden. Demgegenüber war der längenvariable suchbasierte Ansatz in der Lage, den Großteil der Signale zu erzeugen. Dies kann zusätzlich verbessert werden, sobald Wissen, wie z.B. Amplitudenschranken, über das zu erzeugende Signal zur Verfügung steht. Diese mögliche Einschränkung der Signalgenerierung ist eine hauptsächliche Besonderheit des segmentbasierten Ansatzes. Eine gleichartige Einschränkung ist mit dem Polynom-Ansatz nicht möglich.

3.2 BEWERTUNG VON SIGNALEN

In Abschnitt 2.2.2 auf Seite 31 ist der grundlegende Ablauf des suchbasierten Tests beschrieben: Die erstellten Teststimuli werden zunächst zur Ausführung des Testobjekts verwendet. Während der Ausführung wird das Verhalten des Testobjekts mit Hilfe einer geeigneten Instrumentierung beobachtet. Aufgrund der Beobachtungen werden die einzelnen Teststimuli hinsichtlich ihrer Eignung, das Testziel zu erreichen, bewertet.

Der folgende Abschnitt beschreibt eine spezielle Herangehensweise für die Bewertung von in Form von Signalen vorliegenden Teststimuli.

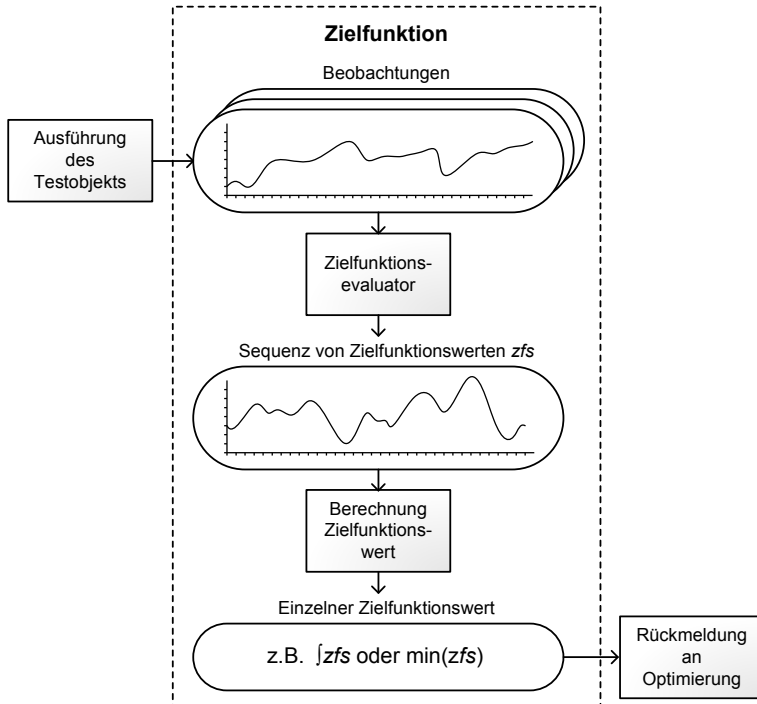


Abb. 29: Berechnung eines einzelnen Zielfunktionswerts bei der Verwendung von Signalen.

ZIELFUNKTIONSWERTBESTIMMUNG

Mit Hilfe der durch die Instrumentierung protokollierten Informationen – gegeben durch mehrere Signalverläufe – muss eine Distanz (Zielfunktionswert) zum Erreichen des jeweiligen Überdeckungsziels berechnet werden. In Abb. 29 sind die dazu notwendigen Schritte illustriert. Die protokollierten Informationen (Beobachtungen) werden für jeden einzelnen Zeitschritt ausgewertet. Der *Zielfunktionsevaluators* berechnet einen Zielfunktionswert für jeden Zeitschritt durch Verwendung spezieller Distanzmetriken basierend auf den Beobachtungen. Es ergibt sich ein spezielles Signal, welches für jeden Zeitschritt die Distanz zur Erreichung des Überdeckungsziels enthält. Je nach Überdeckungsziel können sich die zu verwendenden Metriken und die Anzahl der dazu notwendigen Beobachtungen erheblich unterscheiden. Beispiele

Bewertung

sind die Zielfunktionsberechnungen für den Strukturtest von SL Blöcken oder SF Diagrammen (vgl. Abschnitt 4.2 auf Seite 128).

*Einzelner
Zielfunktionswert*

Metaheuristische Suchverfahren jedoch benötigen zur Bewertung eines Lösungsvorschlags einen einzelnen Zielfunktionswert anstelle einer Sequenz von Zielfunktionswerten. Aus diesem Grund muss aus dieser Sequenz ein einzelner Zielfunktionswert berechnet werden. Die Berechnung des Integrals des Signals, das arithmetische Mittel aller enthaltenen Funktionswerte oder der enthaltene minimale Funktionswert sind sinnvolle Möglichkeiten. Der Vorteil der Verwendung des Integrals oder des arithmetischen Mittels liegt in der Berücksichtigung der Zielfunktionswerte aller Zeitschritte. Somit wird die Suche derart gesteuert, dass sich die Evaluation möglichst zu jedem Zeitpunkt nahe dem Überdeckungsziel befindet. Ist das Überdeckungsziel jedoch in mindestens einem Zeitschritt nicht erfüllt, kann keine Aussage darüber getroffen werden, ob das Überdeckungsziel erfüllt ist. Ein weiterer Nachteil besteht darin, dass sich besonders große und besonders kleine Funktionswerte ausgleichen können und somit nicht in den Optimierungsverlauf einfließen.

Zu diesem Zweck muss der minimale Funktionswert der Zielfunktionswertesequenz zusätzlich in die Zielfunktionswerteberechnung einbezogen werden, da dieser die beste Annäherung an das Überdeckungsziel beschreibt. Der Zielfunktionswert wird in dieser Arbeit wie folgt berechnet:

$$f(zfs) = \begin{cases} 0 & \min(zfs) \leq 0 \\ \overline{zfs} + \min(zfs) & \text{sonst.} \end{cases} \quad (3.10)$$

Wann immer die Sequenz von Zielfunktionswerten zfs mindestens einen Funktionswert enthält, der kleiner oder gleich 0 ist, ist das Überdeckungsziel erreicht, d.h. der optimale Zielfunktionswert 0 wird zurückgegeben. Anderenfalls wird die Summe des arithmetischen Mittels $\overline{zfs}$ aller Funktionswerte von zfs und des darin enthaltenen minimalen Funktionswerts $\min(zfs)$ zurückgegeben.

3.3 SIGNAL-CONSTRAINTS

Der Inhalt dieses Unterkapitels ist in einer im Rahmen dieser Dissertation durchgeführten Diplomarbeit entstanden [148]. Eine alternative Darstellung ist in den Arbeiten von Wilmes und Windisch zu finden [148, 149].

Ein *Constraint* (zu deutsch Nebenbedingung) bezeichnet im Bereich der Testdatengenerierung eine spezielle Anforderung an die zu generierenden Testdaten. Diese gilt zusätzlich zur grundlegenden Anforderung, ein spezifisches Testziel zu erreichen. Derartige Nebenbedingungen können sich aus der Spezifikation des Testobjekts ergeben oder entstammen dem Wissen des Testers. Ein einfaches Beispiel für Constraints eines Systems ist das Vorhandensein von Minimal- bzw. Maximalwerten für seine Eingabeparameter.

Constraints

Die Berücksichtigung von Constraints bei der Generierung von Testdaten ist aus zwei Gründen besonders wichtig: Erstens sind Testdaten, welche die gegebenen Constraints nicht erfüllen, ungültig und für Testzwecke nicht zu gebrauchen. Verwendet beispielsweise ein System im Automobil die Pedalstellung des Beschleunigungs- und des Bremspedals, könnte es etwa nicht erwünscht sein, Testdaten für dieses System zu generieren, welche das gleichzeitige und vollständige Betätigen beider Pedale repräsentieren. Die fehlende Beachtung der Constraints kann jedoch auch dazu führen, dass das Testobjekt bei Ausführung mit diesen Testdaten abstürzt, ohne dass dies notwendigerweise ein Fehlverhalten des Testobjekts darstellt. So können beispielsweise beim Komponententest vorgelagerte Systeme fehlen, welche sonst für die notwendige Plausibilisierung der Daten sorgen würden. Hängt die Funktionsweise eines Systems beispielsweise von der aktuellen Geschwindigkeit des Fahrzeugs ab, so würden Eingaben, welche zu große Sprünge enthalten, d.h. eine zu große Steigung des Geschwindigkeitssignals darstellen, möglicherweise zu einem Fehlverhalten des Systems führen, da es für derart unrealistische Eingaben nicht konzipiert sein könnte. Kann ein Testobjekt aufgrund der fehlenden Beachtung eines Constraints nicht ausgeführt werden, spricht man von einem *kritischen Constraint*. Eine geeignete Berücksichtigung kritischer Constraints ist von besonderer Relevanz, da die Ausführbarkeit der generierten Testdaten eine Voraussetzung für das in dieser Arbeit entwickelte dynamische Testverfahren ist. Der zweite Grund für die Wichtigkeit der Beachtung von Constraints ist die damit verbundene teils

Motivation

starke Einschränkung der Variationsmöglichkeiten der Testdaten. Diese Verkleinerung des Suchraums führt wiederum zu einer Vereinfachung der gesamten Suche. Allein das Einhalten einfacher Constraints, wie beispielsweise Maximal- oder Minimalamplituden, führt zu einer drastischen Verkleinerung des Suchraums. Der reellwertige Wertebereich einer Systemeingabe, welche die Geschwindigkeit eines Fahrzeugs repräsentiert, könnte somit beispielsweise auf den Bereich zwischen 0 und $80 \frac{\text{m}}{\text{s}}$ eingeschränkt werden.

Signal-Constraints

Die beispielhaft erwähnten einfacheren Constraints lassen sich größtenteils mit Hilfe simpler Gleichungssysteme beschreiben. Demgegenüber existieren allerdings auch komplexere Constraints, die Abhängigkeiten unterschiedlicher Systemeingaben über mehrere Zeitschritte hinweg darstellen. Diese für einzelne Signalverläufe definierten Constraints werden als *Signal-Constraints* bezeichnet.

Notwendigkeiten

Um Signal-Constraints im Rahmen eines suchbasierten Verfahrens zur Generierung von Testdaten berücksichtigen zu können, werden zwei elementare Komponenten benötigt: Erstens muss eine ausdrucksstarke Sprache zur Formulierung von Signal-Constraints zur Verfügung stehen, mitsamt der Möglichkeit, einen gegebenen Satz von Signalen bezüglich dieser Constraints auszuwerten. Zweitens muss die Einbeziehung dieser Auswertungen in den Suchprozess unterstützt werden, wodurch die zielgerichtete Generierung Constraint-erfüllender Lösungsvorschläge ermöglicht wird. Dabei muss insbesondere auf die gesonderte Behandlung kritischer Constraints geachtet werden.

Gliederung

Im Folgenden beschreibt Abschnitt 3.3.1 unterschiedliche Möglichkeiten der Spezifikation von Signal-Constraints. Besondere Berücksichtigung finden temporale Logiken, wie die *Signal Temporal Logic*, welche in dieser Arbeit zur Spezifikation von Signal-Constraints verwendet wird. Anschließend geht Abschnitt 3.3.2 auf die Einbeziehung der Signal-Constraints in den zugrundeliegenden Suchprozess ein. Dazu wird in dieser Arbeit eine Penalty Funktion verwendet, welche im Detail vorgestellt wird. Diese Penalty Funktion basiert auf den detaillierten Auswertungen der Lösungsvorschläge bezüglich der definierten Constraints. Diese Auswertungen beschreiben eine numerische Distanz zur Erfüllung des jeweiligen Constraints, d.h. sie drücken die Fähigkeit des Lösungsvorschlags aus, das gegebene Constraint zu erfüllen.

Abschnitt 3.3.3 geht auf die Berechnung dieser Distanzwerte für die verwendete Signal Temporal Logic ein.

3.3.1 TEMPORALE LOGIK FÜR SIGNALE

Zur Spezifikation von Signal-Constraints existiert eine Vielzahl unterschiedlicher Formalismen, wie z.B. reguläre Ausdrücke [6] oder Automaten [91]. Jeder dieser Formalismen hat seine Vor- und Nachteile hinsichtlich der Ausdrucksmächtigkeit und der Einfachheit, notwendige Signaleigenschaften zu spezifizieren.

Eine Klasse dieser Formalismen bilden temporale Logiken, welche die Spezifikation temporaler Eigenschaften ermöglichen. Diese wurden erfolgreich im Bereich der Spezifikation, Verifikation und Validierung reaktiver Systeme unter Verwendung sowohl kontinuierlicher als auch diskreter Signale verwendet [2, 152]. Eine Vielzahl von Arbeiten beschäftigt sich mit der Anpassung temporaler Logiken an die Anforderungen kontinuierlicher Systeme. Aus diesem Grund wird im Folgenden der Fokus auf diesen Formalismus zur Spezifikation von Signal-Constraints gelegt.

*Temporale
Logiken*

Temporale Logiken sind spezielle Vertreter der Modallogik, deren modallogische Operatoren zeitlich interpretiert werden. Sie arbeiten auf einer Struktur, die durch eine Abfolge von Zuständen gegeben ist. Die Operatoren $\Box$ ("always") und $\Diamond$ ("eventually") ermöglichen die Spezifikation von Beziehungen und Abhängigkeiten zwischen den einzelnen Zustandsvariablen an unterschiedlichen Zeitpunkten. Der Operator $\Box$ verlangt die Gültigkeit einer speziellen Formel in allen zukünftigen Zeitschritten bzw. Zuständen, während der Operator $\Diamond$ die Gültigkeit der Formel in mindestens einem zukünftigen Zeitschritt bzw. Zustand fordert. Auf Signale angewandt, entsprechen die Zustände den einzelnen Datenpunkten eines Signals und die Zustandsvariablen entsprechen den zugehörigen Datenwerten.

*Besonder-
heiten*

Die unter Berücksichtigung der spezifischen Anforderungen dieser Arbeit vielversprechendsten Ansätze entstammen dem Bereich der Laufzeitverifikation; dabei werden wie beim Concolic Testing (vgl. Abschnitt 2.2.1 auf Seite 24) formale Verifikationsverfahren und die Ausführung des Testobjekts kombiniert. Die von Maler und Nickovic vorgestellte *Signal Temporal Logic* (STL) [89] ist eine

Varianten

temporale Logik zur Spezifikation von Eigenschaften kontinuierlicher Signale. Die Berücksichtigung kontinuierlicher Signale wird durch deren Transformation in boolesche Signale durch statische Abstraktion ermöglicht. Für das beispielhafte kontinuierliche Signal s_k ergibt die boolesche Abstraktion $s > 5$ ein boolesches Signal, das genau dann wahr ist, wenn der Wert von s_k größer als 5 ist, anderenfalls ist es falsch. Darüber hinaus erlaubt STL die Formulierung von Ereignissen (engl. Events) zur Erkennung von Signalfanken, d.h. zu welchem Zeitpunkt sich der Datenwert eines booleschen Signals ändert. STL basiert auf MITL_[a,b], eine Erweiterung der linearen und kontinuierlichen temporalen Logik *Metric Interval Temporal Logic* (MITL) [3], bei der die temporalen Operatoren auf spezifische zeitliche Intervalle beschränkt werden können. Die Formel $\Box_{[a,b]} s_b$ ist beispielsweise für das gegebene binäre Signal s_b in Zeitpunkt t des Signalverlaufs erfüllt, falls s_b in allen Zeitpunkten des Intervalls $[t + a, t + b]$ wahr ist.

Entscheidung für
STL

STL wurde für diese Arbeit ausgewählt, aufgrund der bereits realisierten Adaption der temporalen Logik an die Anforderungen kontinuierlicher Systeme und die resultierende Berücksichtigung kontinuierlicher Signale in die Constraint-Formulierung und aufgrund des Vorhandenseins systematischer Verfahren zur Auswertung von STL-Formeln [100]. Selbst die Einbeziehung regulärer Ausdrücke ist grundsätzlich möglich [90], soll in dieser Arbeit jedoch nicht berücksichtigt werden.

SIGNAL TEMPORAL LOGIC

Die Syntax und die Semantik der in dieser Arbeit vorgestellten Sprache zur Spezifikation von Signal-Constraints entspricht grundsätzlich jener der STL [90, 92]. Es existieren geringfügige Unterschiede, die im Folgenden betont werden.

Syntax

Abb. 30 gibt eine kompakte Sicht auf die Syntax eines Constraints der verwendeten Sprache. Ein Signal-Constraint besteht aus Operatoren, welche jeweils entweder ein binäres Signal φ oder ein reellwertiges Signal ξ zurückgeben. Viele der Operatoren besitzen bis zu zwei untergeordnete Operatoren. Dadurch können unterschiedliche Operatoren zur Bildung komplexer und hierarchischer Signal-Constraints kombiniert werden, z.B. $\Box((s_1 > 10) \wedge (s_2 < 20))$. Die beiden in die Constraints einbezogenen reellwertigen Signale s_1 und s_2 sind durch den Signal-Operator ξ_s repräsentiert. Der Hauptoperator eines Signal-Constraints, d.h.

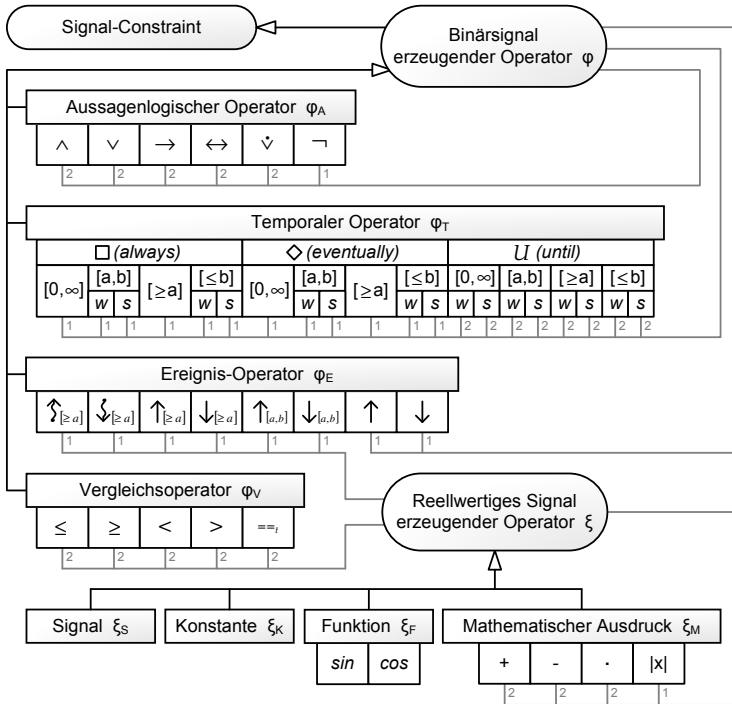


Abb. 30: Syntax eines Signal-Constraints.

der oberste Operator des hierarchischen Aufbaus (im Beispiel der Operator $\Box$), gehört stets zur Klasse der Binärsignale erzeugenden Operatoren.

Im Rahmen der suchbasierten Testdatengenerierung kann eine beliebige Anzahl von Signal-Constraints gegeben sein. Dieser Satz an Constraints ist definiert als die Menge $C = \{\varphi_1, \varphi_2, \dots, \varphi_n\}$, mit $n \in \mathbb{N}$. Jedes der enthaltenen Constraints $\varphi \in O_B = \{\varphi_A, \varphi_T, \varphi_E, \varphi_V\}$ gehört zur Klasse jener Operatoren, die binäre Signale erzeugen. Bei φ handelt es sich entweder um einen aussagenlogischen Operator φ_A , einen Temporaloperator φ_T , einen Ereignis-Operator φ_E oder um einen Vergleichsoperator φ_V . Ein Großteil der Ereignis- und Vergleichsoperatoren besitzen untergeordnete reellwertige Signale erzeugende Operatoren $\xi \in O_R = \{\xi_S, \xi_C, \xi_F, \xi_M\}$, welche entweder durch ein kontinuierliches Signal ξ_S , eine Konstante ξ_K , eine mathematische Funktion ξ_F oder einen mathematischen Operator ξ_M definiert sind.

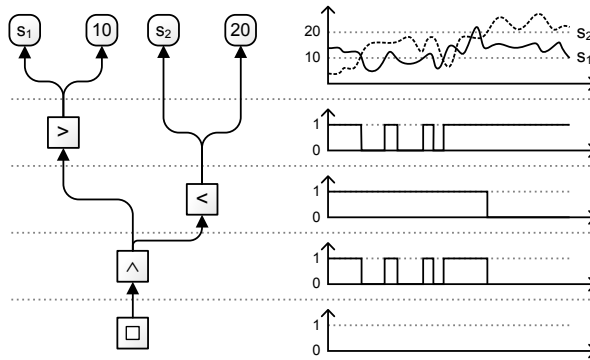


Abb. 31: Semantische Auswertung des Signal Constraints $\varphi = \Box((s_1 > 10) \wedge (s_2 < 20))$ für ein beispielhaftes Individuum i mit den beiden dargestellten Signalen s_1 und s_2 .

Semantik Die Semantik eines Signal-Constraints ist eine hierarchische Verknüpfung der Semantiken jedes enthaltenen Operators. Die Auswertung eines jeden Operators wird für jeden einzelnen Datenpunkt $d \in [0, D]$ durchgeführt. Die Semantik eines Binärsignale erzeugenden Operators φ kann durch die Funktion $b_d : O_B \times P \rightarrow \mathbb{B}$ beschrieben werden, die Semantik eines reellwertige Signale erzeugenden Operators ξ , durch $r_d : O_R \times P \rightarrow \mathbb{R}$.

Das von einem Operator φ für Individuum i erzeugte Binärsignal b bildet ab, an welchem Datenpunkt d die durch φ repräsentierte Eigenschaft erfüllt ist ($b_d(\varphi, i) = 1$) oder verletzt wird ($b_d(\varphi, i) = 0$). Ist φ der Hauptoperator des Constraints und gilt $\forall d \in [0, D] : b_d(\varphi, i) = 1$, d.h. ist die Eigenschaft zu jedem Zeitpunkt im Signalverlauf erfüllt, so wird das Constraint als erfüllt angesehen. In diesem Fall wird das Individuum i als ein *gültiges Individuum* bezeichnet, anderenfalls als ein *ungültiges Individuum*.

Beispiel Eine beispielhafte semantische Auswertung des zuvor erwähnten Signal-Constraints $\Box((s_1 > 10) \wedge (s_2 < 20))$ ist in Abb. 31 illustriert. Auf der linken Seite ist die syntaktische Struktur des Constraints zu sehen. Auf der rechten Seite sind zunächst die beiden reellwertigen Signale s_1 und s_2 des beispielhaften Individuums abgebildet. Darunter befindet sich für jeden Operator des Constraints jeweils das resultierende Binärsignal b . Wie dargestellt, berechnet der $>$ Operator für jeden Datenpunkt $d \in [0, D]$, ob der Signalwert von s_1 kleiner als die Konstante 10 ist. Analog verhält sich der $<$ Operator bzgl. Signal s_2 und der Konstanten 20.

Die jeweils resultierenden Binärsignale bilden nun die Grundlage für die Berechnung des Operators $\wedge$ (logisches *UND*). Das daraus resultierende Signal ist wiederum Grundlage für den Hauptoperator $\square$ des Constraints, welcher für jeden Datenpunkt berechnet, ob alle folgenden Datenpunkte wahr sind. Dies ist in diesem Beispiel nicht der Fall, d.h. das Constraint ist für das gegebene Individuum bzw. für die darin enthaltenen Signale s_1 und s_2 nicht gültig; i ist somit ein ungültiges Individuum.

OPERATOREN DER SIGNAL TEMPORAL LOGIC

Die Syntax und Semantik nahezu aller in Abb. 30 aufgeführten Operatoren ist von Maler und Nickovic formal beschrieben worden [89]. Im Folgenden soll aus diesem Grund nur kurz auf die Funktionsweise der Operatoren eingegangen werden. Zusätzlich werden die für diese Arbeit hinzugefügten bzw. angepassten Operatoren hervorgehoben.

Reellwertige Signale erzeugende Operatoren ($\xi \in O_R$)

Für ein Individuum i erzeugen diese Operatoren ein reellwertiges Signal, welches für jeden Datenpunkt $d \in [0, D)$ mit Hilfe der Funktion $r_d(\xi, i)$, mit $\xi \in O_R$ beschrieben wird. Diese erzeugten reellwertigen Signale werden überwiegend zu Referenz- oder Vergleichszwecken verwendet. Im Gegensatz zu Binärsignalen erzeugenden Operatoren treffen diese keine Aussagen.

Der *Signal-Operator* ξ_S beispielsweise referenziert direkt die im Rahmen des suchbasierten Tests generierten Signale. Konstante Signale, die für jeden Zeitschritt einen konstanten Datenwert $k \in \mathbb{R}$ aufweisen, werden von einem *Konstanten-Operator* ξ_K erzeugt. Konstante Signale werden üblicherweise zu Vergleichszwecken verwendet, z.B. zur Definition spezifischer Schranken bzw. Schwellwerte. *Mathematische Operatoren* ξ_M erzeugen ein reellwertiges Signal durch Anwendung mathematischer Rechenoperatoren aus den reellwertigen Signalen eines oder zweier untergeordneter Operatoren. Es werden die Operatoren Addition, Subtraktion, Multiplikation und die Bestimmung von Absolutwerten unterstützt.

Funktionsoperatoren ξ_F erstellen ein reellwertiges Signal basierend auf einer mathematischen Funktion. Funktionsoperatoren sind kein Bestandteil der von Maler und Nickovic vorgestellten Signal Temporal Logic. Sie wurden im Rahmen dieser Arbeit entwickelt,

da es möglich sein kann, dass eine spezifische Signaleigenschaft auf einer solchen mathematischen Funktion basiert, die daher entsprechend berücksichtigt werden muss. Vorerst werden lediglich die Funktionen Sinus und Kosinus berücksichtigt. Syntaktisch sind die Funktionsoperatoren durch den Namen der definierten mathematischen Funktionen gegeben: $\xi_F \in \{\sin, \cos\}$. Das von einem Funktionsoperator erzeugte reellwertige Signal setzt sich aus den Funktionswerten der assoziierten Funktion zusammen. Für jeden Datenpunkt d wird der Funktionswert zum Zeitpunkt $d \cdot R$ berechnet:

$$r_d(\xi_F, i) = f(d \cdot R), f \in \{\sin, \cos\}. \quad (3.11)$$

Binärsignale erzeugende Operatoren ($\varphi \in O_B$)

Diese Operatoren erzeugen für jedes Individuum i ein Binärsignal, welches für jeden Datenpunkt $d \in [0, D)$ mit Hilfe der Funktion $b_d(\varphi, i)$, mit $\varphi \in O_B$ beschrieben wird. Diese Binärsignale beschreiben spezifische Signaleigenschaften. Jeder Datenwert entspricht einem Wahrheitswert, welcher Auskunft über die Gültigkeit dieser Eigenschaft gibt.

- φ_V *Vergleichsoperatoren* φ_V setzen die reellwertigen Signale der untergeordneten Operatoren $\xi \in O_R$ vergleichend in Relation. Zusätzlich zu den von Maler und Nickovic beschriebenen Operatoren ($>, \geq, <, \leq$) wird in dieser Arbeit zusätzlich ein Gleichheitsoperator eingeführt, da Signalvergleiche, z.B. mit konstanten Signalen, häufig in Signal-Constraints vorkommen. Dieser Gleichheitsoperator besitzt zudem einen parametrierbaren Toleranzwert $t \in \mathbb{R}$, durch welchen die Abfrage der Ähnlichkeit unterschiedlicher Signale möglich ist. Das Signal wird in diesem Fall ($\varphi_V \hat{=} \xi_1 \stackrel{t}{=} \xi_2$) für jeden Datenpunkt durch die Auswertung der Aussage $|r_d(\xi_1, i) - r_d(\xi_2, i)| \leq t$ erzeugt.
- φ_A Mit Hilfe *Aussagenlogischer Operatoren* (φ_A) lassen sich zwei Aussagen in eine logische Beziehung setzen bzw. eine einzelne Aussage negieren. Aussagen sind in Form von Binärsignalen der untergeordneten Operatoren gegeben. Unterstützt werden die aussagenlogischen Operatoren Konjunktion, Disjunktion, Äquivalenz, Kontravalenz und Negation.
- φ_T *Temporale Operatoren* (φ_T) treffen für jeden Datenpunkt d Aussagen über die zeitliche Gültigkeit von Signaleigenschaften. In der von Maler und Nickovic vorgestellten Signal Temporal Logic

existieren Zukunfts- und Vergangenheitsoperatoren. Da letztere die Ausdrucksmächtigkeit einer temporalen Logik nicht erweitern [90, 142], werden sie in dieser Arbeit nicht berücksichtigt. Unterstützt werden die temporalen Zukunftsoperatoren „always“ ($\Box$), „eventually“ ($\Diamond$) und „until“ ($\mathcal{U}$). Wie eingangs erwähnt, basiert die STL auf $\text{MITL}_{[a,b]}$. Zur Definition detaillierterer temporaler Eigenschaften ist daher für jeden dieser Operatoren die Spezifikation eines Gültigkeitsintervalls möglich. STL unterscheidet darüber hinaus in strikte (engl. strong) (s) und schwache (engl. weak) (w) Varianten der temporalen Operatoren, was Einfluss auf den semantischen Umgang mit den Intervallangaben hat [100].

Demgegenüber bestimmen *Ereignis-Operatoren* (φ_E) die Zeitpunkte, an denen eine spezifische Signaleigenschaft beginnt oder endet. Die Ereignis-Operatoren *Rise* ($\uparrow$) und *Fall* ($\downarrow$) geben genau dann wahr zurück, wenn das Binärsignal des untergeordneten Operators zu wahr bzw. zu falsch wechselt. Für diese Arbeit wurde die temporale Logik um Operatoren erweitert, welche den Anstieg oder das Fallen reellwertiger Signale erkennen. Syntaktisch sind diese beschrieben als $\varphi_E \in \{\uparrow_{[\geq a]} \xi, \downarrow_{[\geq a]} \xi, \uparrow_{[\geq a]} \xi, \downarrow_{[\geq a]} \xi, \uparrow_{[a,b]} \xi, \downarrow_{[a,b]} \xi\}$. Die *strikten* Rise Operatoren ($\uparrow$) überprüfen, ob das durch sie evaluierte reellwertige Signal für alle Datenpunkte des betrachteten Intervalls stetig steigt. Für die *schwachen* Rise Operatoren ($\uparrow$) muss der Signalanstieg demgegenüber nicht stetig sein. Die strikten und *schwachen* Fall Operatoren ($\downarrow$ bzw. $\downarrow$) fordern analog ein stetiges bzw. nicht-stetiges Fallen des Signals.

 φ_E

3.3.2 BERÜCKSICHTIGUNG WÄHREND DER SUCHE

Die Arbeiten von Coello [28] und Michalewicz und Schoenauer [96, 97] geben einen umfassenden Überblick über verschiedene Möglichkeiten, Constraints bei der Lösung von Optimierungsproblemen zu berücksichtigen. Der am weitesten verbreitete Ansatz ist die Verwendung von *Penalty* Funktionen [32].

Die Grundidee bei der Verwendung einer *Penalty* Funktion besteht darin, bei der Bewertung der Lösungsvorschläge einen zusätzlichen Strafwert (engl. *Penalty*) in die Fitness einfließen zu lassen. Dieser Strafwert bemisst das Ausmaß der Verletzung der Constraints. Bei Erfüllung der Constraints, ergibt sich eine Verletzung von Null, anderenfalls eine Verletzung größer Null. Diese Einbeziehung eines Strafwertes in die Fitnessbewertung der

*Penalty
Funktionen*

Lösungsvorschläge kombiniert die Verfolgung des eigentlichen Optimierungsziels mit dem Ziel, die definierten Constraints zu erfüllen.

Vorteile Die Verwendung von Penalty Funktionen zur Berücksichtigung von Signal-Constraints, wie sie mit Hilfe der Signal Temporal Logic spezifiziert werden können, ist aus den folgenden Gründen besonders erfolgversprechend:

- A. Der Ansatz ist unabhängig vom verwendeten Optimierungsverfahren. Die einzige Voraussetzung ist, dass es sich um ein metaheuristisches Suchverfahren handelt.
- B. Eine beliebige Form der Definition der Constraints kann verwendet werden, so auch die Signal Temporal Logic. Die einzige Bedingung ist die Möglichkeit, ein Maß für die Constraintverletzung angeben zu können.
- C. Der Ansatz ist ebenfalls zielführend, wenn zu Beginn der Optimierung kein einziges Individuum das Constraint erfüllt.

Herausforderungen Die Bemessung der Strafwerte ist eine der größten Herausforderungen bei der Verwendung von Penalty Funktionen. Wird dieser Strafwert im Vergleich zum bereits vorhandenen Fitnesswert zu groß bemessen, so wird die Erfüllung der Constraints während der Suche höher priorisiert, als das Erreichen des zugrundeliegenden Optimierungsziels. Dies führt dazu, dass die Constraints im Optimierungsverlauf sehr schnell erfüllt sind, das Optimierungsziel jedoch möglicherweise nicht erreicht werden kann, da die Suche aufgrund der fehlenden Diversität der Population in einem lokalen Optimum endet. Werden die Strafwerte demgegenüber zu klein bemessen, werden die definierten Constraints nicht genügend berücksichtigt.

Kategorien Je nach Bemessung der Strafwerte lassen sich Penalty Funktionen in zwei grundlegende Kategorien unterscheiden: statische und dynamische Penalty Funktionen. Bei den *statischen* Penalty Funktionen wird der berechnete Strafwerte gewichtet auf die bereits berechnete Fitness addiert. Die Bemessung der Strafmaße und damit der Erfolg der Suche hängt in diesem Fall sehr stark von der gewählten Gewichtung ab. *Dynamische* Penalty Funktionen beziehen zusätzlich Informationen über den Optimierungsverlauf in die Bemessung der Strafwerte ein. So steigt üblicherweise die Gewichtung der Strafwerte mit Fortschreiten der Optimierung an.

Zu Beginn der Suche wird so besonderer Wert auf das Erreichen des Optimierungsziels und eine ausgiebige Erkundung des Suchraums gelegt, wohingegen die Einhaltung der Constraints zum Ende der Suche hin priorisiert wird. Dieses Vorgehen führt im Vergleich zu statischen Penalty Funktionen zu besseren Ergebnissen [28].

Eine besondere Variante dynamischer Penalty Funktionen sind *adaptive* Penalty Funktionen. Diese nutzen zur Bemessung der Strafwerte Informationen über bestimmte Eigenschaften der jeweils aktuellen Population. Bei der von Tessema und Yen vorgestellten Variante wird der Anteil gültiger Individuen an der Gesamtanzahl der Individuen innerhalb der Population zur Gewichtung der Strafwerte verwendet [124]. Je geringer dieser Anteil ist, desto stärker werden die Strafwerte gewichtet, d.h. desto stärker wird nach gültigen Individuen gesucht. Befinden sich hingegen viele gültige Individuen in der aktuellen Population, so wird die Suche verstärkt bzgl. des Auffindens einer Lösung des Optimierungsproblems ausgerichtet. Ein Vorteil des Verfahrens ist, dass in beiden Fällen jeweils ein gewisser Anteil suboptimaler Individuen erhalten bleibt (sowohl bzgl. des Optimierungsziels als auch bzgl. der Erfüllung der Constraints), was zu einer Erhöhung der Robustheit der Suche gegenüber lokaler Optima führt. Durch die selbständige Adaption der Gewichtung entfällt darüber hinaus die Notwendigkeit der Spezifikation problemspezifischer Parameter und die damit verbundene Feinabstimmung. Des weiteren ist die Implementation dieser adaptiven Penalty Funktion sehr einfach und sie erzeugt keinen hohen zusätzlichen Rechenaufwand. Tessema und Yen haben gezeigt, dass sie mindestens ebenso gute Ergebnisse erzielt, wie andere aktuelle Verfahren [124]. Aus diesen Gründen wird diese Penalty Funktion in dieser Arbeit zur Berücksichtigung der Signal Constraints während der Suche verwendet.

*Adaptive
Penalty
Funktion*

Im Folgenden wird beschrieben, wie diese Penalty Funktion in den Suchprozess der suchbasierten Testdatengenerierung eingebettet ist und welche Erweiterungen für diesen Einsatz notwendig sind.

AUFNAHME DER PENALTY FUNKTION IN DEN SUCHPROZESS

Der Ablauf des suchbasierten Tests unter Einbeziehung einer Penalty Funktion zur Berücksichtigung von Constraints ist in Abb. 32 illustriert. Das Suchverfahren schlägt mit jedem Individuum $i \in P$

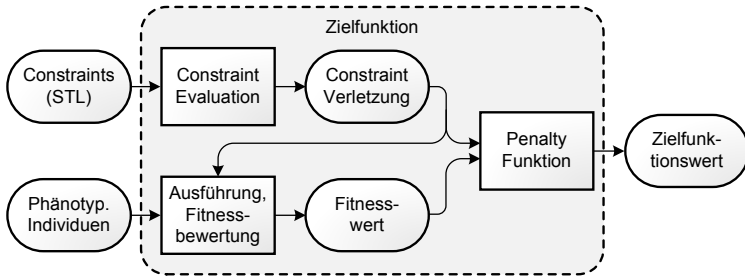


Abb. 32: Berücksichtigung der Signal Constraints bei der Berechnung der Zielfunktionswerte durch Einbettung einer Penalty Funktion.

einen Satz von Signalen vor (vgl. Abb. 12 auf Seite 51), welche bzgl. der Einhaltung der definierten Constraints evaluiert werden. Die Constraints liegen in Form der Signal Temporal Logik vor (vgl. Absatz 3.3.1). Aufgrund dieser Evaluation wird – falls die Constraints nicht erfüllt sind – ein Distanzmaß $v(i) \in \mathbb{R}_{[0,1]}$ berechnet, welches Aufschluss über das Ausmaß der Constraint Verletzung gibt (diese Distanzmaße werden ausführlich in Abschnitt 3.3.3 beschrieben). Bei der Einbeziehung der Constraint Verletzungen wird unterschieden in die Menge der Individuen, die mindestens ein kritisches Constraint verletzen ($P^- \subseteq P$) und jene, die kein kritisches Constraint verletzen ($P^+ \subseteq P$). Alle Individuen der Menge P^+ können ausgeführt und zur Berechnung eines Fitnesswertes $f(i)$ verwendet werden. Demgegenüber dürfen die Individuen aus P^- nicht ausgeführt werden. Für diese existiert somit auch kein Fitnesswert. Basierend auf dem Fitnesswert $f(i)$ (falls $i \in P^+$) und der Constraint Verletzung $v(i)$ berechnet die Penalty Funktion im Anschluss einen endgültigen Zielfunktionswert $F(i)$. Ist kein Fitnesswert vorhanden (falls $i \in P^-$), berechnet sich der Zielfunktionswert einzig basierend auf der Constraint Verletzung.

Im Wesentlichen handelt es sich bei der Penalty Funktion um die von Tessema und Yen vorgestellte adaptive Penalty Funktion [124]. Erweitert wurde diese für diese Arbeit bezüglich der Unterscheidung in kritische und nicht-kritische Constraints.

Skalierung
der
Fitnesswerte

Bei den für alle Individuen $i \in P^+$ berechneten Fitnesswerten $f(i)$ kann es sich abhängig von der verwendeten Fitnessfunktion um beliebige reellwertige Zahlen handeln. Damit diese mit den berechneten Constraint Verletzungen $v(i) \in [0, 1]$ kombiniert werden

können, müssen sie zuvor ebenfalls auf das Intervall $[0, 1]$ skaliert werden. Dies geschieht durch die Funktion $f_S : P^+ \rightarrow \mathbb{R}_{[0,1]}$:

$$f_S(i) = \frac{f(i)}{\max_{j \in P^+} f(j)}. \quad (3.12)$$

Das schlechteste Individuum der Population erhält den Fitnesswert 1, das beste bekommt einen entsprechend geringeren Fitnesswert zugewiesen. Den global optimalen Fitnesswert 0 erhalten nur jene Individuen, welche diesen Fitnesswert bereits vor der Skalierung besitzen.

Wie eingangs beschrieben, verwendet die adaptive Penalty Funktion von Tessema und Yen Eigenschaften der aktuellen Population. Es handelt es sich um den Anteil $\alpha_g \in [0, 1]$ gültiger Individuen an der gesamten Population, welche sich wie folgt berechnet:

*Gültigkeit
der
Population*

$$\alpha_g = \frac{|\{i \in P \mid v(i) = 0\}|}{|P|}. \quad (3.13)$$

Für alle Individuen der Population P muss ein endgültiger Zielfunktionswert $F(i)$ berechnet werden. Für alle Individuen, welche keine kritischen Constraints verletzen, berechnet sich der endgültige Zielfunktionswert wie folgt: $\forall i \in P^+ :$

*Zielfunktionswert
 $i \in P^+$*

$$F(i) = \begin{cases} f_S(i) , & \text{falls } v(i) = 0 \\ v(i) , & \text{falls } \alpha_g = 0 \\ \sqrt{f_S(i)^2 + v(i)^2} + (1 - \alpha_g) \cdot v(i) + \alpha_g \cdot f_S(i) , & \text{sonst.} \end{cases} \quad (3.14)$$

Handelt es sich bei Individuum i um ein gültiges Individuum ($v(i) = 0$), wird direkt der für i berechnete Fitnesswert $f(i)$ als Zielfunktionswert zurückgegeben. Somit wird die Fitness in diesem Fall (bis auf die Skalierung auf das Intervall $[0, 1]$) nicht verfälscht [124]. Befindet sich hingegen kein einziges gültiges Individuum in der aktuellen Population, wird für jedes Individuum i die Constraint Verletzung $v(i)$ als Zielfunktionswert zurückgegeben. Dies führt dazu, dass gerade zu Beginn der Suche, wo üblicherweise keines der Individuen die Constraints erfüllt, besonderen Wert auf das Erfüllen der Constraints gelegt wird. In allen anderen Fällen, d.h. wenn es sich bei i um ein ungültiges Individuum handelt, gleichzeitig allerdings gültige Individuen

innerhalb der aktuellen Population existieren, berechnet sich der Zielfunktionswert als eine Kombination der beiden Distanzmaße $f(i)$ und $v(i)$.

Zielfunktionswert
 $i \in P^-$

Für alle Individuen, welche mindestens ein kritisches Constraint verletzen, steht kein Fitnesswert $f(i)$ zur Verfügung. Der Zielfunktionswert muss in diesen Fällen ausschließlich basierend auf der Constraint Verletzung der Individuen berechnet werden. Es muss gleichzeitig dafür Sorge getragen werden, dass die Individuen aus P^+ besser gestellt werden als jene aus P^- , d.h. die berechneten Zielfunktionswerte der Individuen aus P^- schlechter sind, als der jeweils schlechteste Zielfunktionswert aller Individuen aus P^+ . Der Zielfunktionswert berechnet sich wie folgt: $\forall i \in P^-$:

$$F(i) = \begin{cases} v(i) & , \text{ falls } P^+ = \emptyset \\ \left[\max_{j \in P^+} F(j) \right] + v(i) & , \text{ sonst} \end{cases} \quad (3.15)$$

Falls alle Individuen der Population P mindestens ein kritisches Constraint verletzen ($P^+ = \emptyset$), wird die Constraint Verletzung als Zielfunktionswert zurückgegeben. Dies führt zu einer ausschließlichen Suche nach gültigen Individuen. Anderenfalls entspricht der Zielfunktionswert der Summe der Constraint Verletzung und des schlechtesten Zielfunktionswertes aller Individuen aus P^+ .

3.3.3 DISTANZMASSE

Die Verwendung einer Penalty-Funktion zur Berücksichtigung von Signal-Constraints in einem Suchverfahren erfordert eine geeignete Bemessung des Grades, mit dem die Individuen i einer Population P die untersuchten Constraints C verletzen. Es soll durch einen numerischen Wert beschrieben werden, inwiefern ein Individuum zur Erfüllung der Constraints geeignet ist. Wie in Abb. 33 beispielhaft illustriert, wird der Suchraum durch die Einbeziehung von Constraints im Suchverfahren in Regionen gültiger und ungültiger Individuen unterteilt. Für jedes Individuum i ist die Constraint-Verletzung $v : P \rightarrow \mathbb{R}_{[0,1]}$ gegeben als seine normierte Distanz zur nächstgelegenen Region gültiger Individuen. Den maximal möglichen Wert 1 erhält das Individuum, welches im Vergleich aller anderen Individuen der Population die Constraints C „am stärksten“ verletzt. Im Folgenden wird genauer auf die Berechnung dieser Constraint-Verletzung eingegangen.

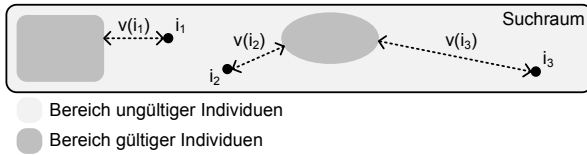


Abb. 33: Partitionierung des Suchraums in Bereiche gültiger und ungültiger Individuen. Die Differenz der drei Individuen i_1 , i_2 und i_3 zu gültigen Bereichen ist mit dem Verletzungsmaß v angegeben.

CONSTRAINT-VERLETZUNG ALS DISTANZMASS

Die durch die Semantik der Signal-Constraints gegebene Auswertungsmöglichkeit reicht für eine Bemessung der Constraint-Verletzung nicht aus. Es muss stattdessen ein detailliertes Distanzmaß für die definierten Constraints C verwendet werden, welches die Distanz zur Erfüllung der Constraints angibt.

Da mehrere Constraints definiert sein können, muss jeweils die Distanz v_C (Constraint-Verletzung) des evaluierten Individuums $i \in P$ zu jedem dieser definierten Constraints $c \in C$ berechnet werden. Die Funktion $\tilde{v} : P \rightarrow \mathbb{R}$ berechnet das gewichtete Mittel dieser einzelnen Constraint-Verletzungen. Jedem Signal-Constraint ist eine feste Gewichtung zugeordnet: $w : C \rightarrow \mathbb{R}_{>0}$. Der durch die Funktion $\tilde{v}$ berechnete Wert der gesamtheitlichen Constraint-Verletzung muss für die Anwendung einer Penalty-Funktion auf das Intervall $[0, 1]$ skaliert werden. Zu diesem Zweck berechnet die Funktion v für jedes Individuum i den Quotienten aus dessen Constraint-Verletzung und der maximalen Constraint-Verletzung aller Individuen aus dessen Population.

Definition 3.1. Die *Gesamt-Constraint-Verletzung* bemisst die Verletzung aller definierten Constraints $c \in C$ für ein Individuum i . Sie ist eine Funktion $v : P \rightarrow \mathbb{R}_{[0,1]}$ und ist definiert als das normalisierte arithmetische Mittel der Constraint-Verletzungen aller enthaltenen Constraints:

$$v(i) = \frac{\tilde{v}(i)}{\max_{j \in P} \tilde{v}(j)}, \tilde{v}(i) = \frac{\sum_{x=0}^{|C|} w(\varphi_x) \cdot v_C(i, \varphi_x)}{\sum_{x=0}^{|C|} w(\varphi_x)}. \quad (3.16)$$

Zur Berechnung der Constraint-Verletzung eines Individuums i bzgl. eines einzelnen Constraints wird das Binärsignal $b(\varphi, i)$

verwendet, welches vom Haupt-Operator φ des Constraints erzeugt wird (vgl. Absatz 3.3.1 auf Seite 94). Eine erste Idee ist es, den Anteil aller falsch-Werte an der Gesamtanzahl D der Signalewerte als Constraint-Verletzungsmaß zu verwenden. Dieses Distanzmaß wird als *Constraint-Verletzungsrate* bezeichnet. Die Verletzungsrate allein reicht jedoch für die Bemessung der Constraint-Verletzung nicht aus, da unterschiedliche Individuen, welche das Constraint jeweils in jedem Zeitschritt verletzen, nicht unterscheidbar sind. Aus diesem Grund wird in dieser Arbeit zur Berechnung der Constraint-Verletzung eines Individuums i bzgl. des Constraints φ zusätzlich zu v_R ein weiteres Distanzmaß verwendet: das *Constraint-Verletzungsmaß* v_A .

Definition 3.2. Die *Constraint-Verletzung* ist ein Maß für die Verletzung eines einzelnen Constraints $c \in C$ für ein Individuum i . Sie ist eine Funktion $v_C : P \times C \rightarrow \mathbb{R}$ und ist definiert als die Summe der gewichteten Constraint-Verletzungsrate und des Constraint-Verletzungsmaßes:

$$v_C(i, \varphi) = \alpha \cdot v_R(i, \varphi) + v_A(i, \varphi). \quad (3.17)$$

Das bereits motivierte Distanzmaß v_R nutzt lediglich die in Form eines Binärsignals gegebene Auswertung des Haupt-Operators φ des evaluierten Constraints. Die booleschen Werte wahr und falsch werden dabei als die Zahlenwerte 0 und 1 aufgefasst. Die Constraint-Verletzungsrate gibt somit eine grobe erste Bemessung der Constraint-Verletzung.

Definition 3.3. Die *Constraint-Verletzungsrate* ist ein Summand der Constraint-Verletzung v_C und realisiert ein grobes Maß für die Verletzung eines Constraints $c \in C$ für ein Individuum i . Sie ist eine Funktion $v_R : P \times O_B \rightarrow \mathbb{R}_{[0,1]}$ und ist definiert als der Anteil an falsch-Werten des vom Haupt-Operator des Constraints erzeugten Binärsignals an allen Datenpunkten:

$$v_R(i, \varphi) = \frac{1}{D} \cdot \sum_{x=0}^{D-1} 1 - b_x(\varphi, i). \quad (3.18)$$

Als Ergänzung zu v_R liefert das Constraint-Verletzungsmaß eine feinere Bemessung der Constraint-Verletzung, da dadurch berechnet wird, inwieweit das jeweilige Constraint verletzt wird. Zu diesem Zweck muss die Semantik der Binärsignale erzeugenden Operatoren $\varphi \in O_B$ dahingehend angepasst werden, dass

zusätzlich zum erzeugten Binärsignal b zwei weitere reellwertige Signale δ^e und δ^v erzeugt werden, welche die Distanz zur Erfüllung bzw. zur Verletzung des Constraints angeben. Die beiden Signale $\delta^e, \delta^v : O_B \times P \rightarrow \mathbb{R}_{[0,1]}$ berechnen für jeden Datenpunkt $d \in [0, D - 1]$ einen positiven Distanzwert im Intervall $[0, 1]$. Die Signale δ^e und δ^v werden im Folgenden für jeden Operator φ spezifisch definiert.

Für das Constraint-Verletzungsausmaß wird die Distanz zur Erfüllung (Signal δ^e) des Haupt-Operators des evaluierten Signal-Constraints verwendet. Die Datenwerte δ_d^e aller Datenpunkte von δ^e werden aufaddiert und anschließend in das Intervall $[0, 1]$ skaliert. Dazu wird die jeweils berechnete Summe durch die maximale berechnete Summe aller Individuen der aktuellen Population verwendet.

Definition 3.4. *Das Constraint-Verletzungsausmaß ist der zweite Summand der Constraint-Verletzung v_C . Es ist eine Funktion $v_A : P \times O_B \rightarrow \mathbb{R}_{[0,1]}$ und ist definiert als die Distanz des evaluierten Individuums i zur Erfüllung des Constraint-Operators φ in jedem Datenpunkt $d \in [0, D - 1]$:*

$$v_A(i, \varphi) = \begin{cases} 0, & \text{falls } v_R(i, \varphi) = 0 \\ \frac{\tilde{v}_A(i, \varphi)}{\max_{j \in P} \tilde{v}_A(j, \varphi)}, & \text{sonst} \end{cases}$$

$$\tilde{v}_A(i, \varphi) = \sum_{d=0}^{D-1} \delta_d^e(\varphi, i). \quad (3.19)$$

OPERATOREN-SPEZIFISCHE DISTANZMASSE

Dieser Abschnitt definiert die einzelnen Operatoren-spezifischen Distanzmaße δ^e und δ^v für ein Individuum i und einen Operator $\varphi \in O_B$. Es werden lediglich Binärsignale erzeugende Operatoren betrachtet, da nur diese Haupt-Operatoren eines Constraints sein können.

Die unterschiedlichen Operatoren können unterschiedliche Wertebereiche aufweisen und sind somit nicht direkt vergleichbar. Aus diesem Grund werden die Signalwerte der Signale δ^e und δ^v auf das Intervall $[0, 1]$ skaliert. Im Folgenden stellen die Funktionen $\tilde{\delta}^e$ und $\tilde{\delta}^v$ die nicht-skalierten Varianten von δ^e bzw. δ^v dar. Die

Skalierung wird ebenfalls durch Division durch den maximalen Datenwert aller Datenpunkte aller Individuen der Population durchgeführt.

Die Gleichung 3.20 beschreibt die zugrundeliegende Berechnung des Signals δ^e für jeden Datenpunkt $d \in [0, D - 1]$. Die Distanz eines Individuums i zur Erfüllung eines Constraints $\varphi \in O_B$ an einem bestimmten Datenpunkt d ist genau dann 0, falls das von dessen Hauptoperator erzeugte Binärsignal an dieser Stelle ($b_d(\varphi, i)$) 1 ist, d.h. wo das Constraint erfüllt ist. Anderenfalls berechnet sich diese Distanz durch Skalierung des Operator-spezifischen Distanzwertes $\tilde{\delta}^e(\varphi, i)$ für den jeweiligen Operator $\varphi \in O_B$.

$$\delta_d^e(\varphi, i) = \begin{cases} 0 & , \text{ falls } b_d(\varphi, i) = 1 \\ \frac{\tilde{\delta}_d^e(\varphi, i)}{\max_{\substack{j \in P \\ t \in [0, D-1]}} \tilde{\delta}_t^e(\varphi, j)} & , \text{ sonst} \end{cases} \quad (3.20)$$

Die in Gleichung 3.21 beschriebene Distanz eines Individuums i zur Verletzung eines Constraints berechnet sich analog: δ^v ist genau dann 0, falls das Constraint nicht erfüllt ist. Anderenfalls wird das entsprechende Distanzmaß $\tilde{\delta}^v$ wie beschrieben skaliert.

$$\delta_d^v(\varphi, i) = \begin{cases} 0 & , \text{ falls } b_d(\varphi, i) = 0 \\ \frac{\tilde{\delta}_d^v(\varphi, i)}{\max_{\substack{j \in P \\ t \in [0, D-1]}} \tilde{\delta}_t^v(\varphi, j)} & , \text{ sonst} \end{cases} \quad (3.21)$$

Die folgenden Abschnitte gehen nun auf die Berechnung der unskalierten Distanzmaße $\tilde{\delta}^e$ und $\tilde{\delta}^v$ für die vier Binärsignale erzeugenden Operatoren φ_V , φ_A , φ_T und φ_E ein (vgl. Absatz 3.3.1 auf Seite 94). Es wird jeweils die Berechnung für einen Datenpunkt d beschrieben.

$\tilde{\delta}^e$ und $\tilde{\delta}^v$ für Vergleichsoperatoren φ_V

Die Vergleichsoperatoren setzen jeweils die beiden von den untergeordneten Operatoren ξ_1 und ξ_2 erzeugten reellwertigen Signale $r(\xi_1, i)$ und $r(\xi_2, i)$ in Relation. Als Grundlage für die Berechnung der Distanz zur Erfüllung und der Distanz zur Verletzung des definierten Constraints dient jeweils die Distanz zwischen den

φ_V	$\tilde{\delta}_d^e(\varphi_V, i)$	$\tilde{\delta}_d^v(\varphi_V, i)$
$\xi_1 ==_t \xi_2$	$\delta_{==}(\xi_d^1, \xi_d^2) - t$	$-1 \cdot \tilde{\delta}_d^e(\xi_1 ==_t \xi_2, i)$
$\xi_1 > \xi_2$	$\delta_{>}(\xi_d^1, \xi_d^2)$	$\tilde{\delta}_d^e(\xi_1 \leq \xi_2, i)$
$\xi_1 < \xi_2$	$\delta_{<}(\xi_d^1, \xi_d^2)$	$\tilde{\delta}_d^e(\xi_1 \geq \xi_2, i)$
$\xi_1 \geq \xi_2$	$\delta_{\geq}(\xi_d^1, \xi_d^2)$	$\tilde{\delta}_d^e(\xi_1 < \xi_2, i)$
$\xi_1 \leq \xi_2$	$\delta_{\leq}(\xi_d^1, \xi_d^2)$	$\tilde{\delta}_d^e(\xi_1 > \xi_2, i)$
dabei gilt $\xi_d^1 \triangleq r_d(\xi_1, i)$, $\xi_d^2 \triangleq r_d(\xi_2, i)$		

Tab. 3: Berechnung der Distanzmaße $\tilde{\delta}^e$ und $\tilde{\delta}^v$ für die Vergleichsoperatoren $\varphi_V \in O_B$ und ein gegebenes Individuum i .

einzelnen Datenwerten der Signale $r(\xi_1, i)$ und $r(\xi_2, i)$. Dazu können die vom suchbasierten Strukturtest bekannten Distanzmaße für Relationsoperatoren ($\delta_{==}$, $\delta_{>}$, $\delta_{<}$, $\delta_{\geq}$, $\delta_{\leq}$) verwendet werden (vgl. Tab. 1 auf Seite 37).

Tab. 3 fasst die Berechnung der einzelnen Distanzmaße zusammen. Diese stimmen bis auf den Operator $\varphi_V \triangleq (\xi_1 ==_t \xi_2)$ mit den bereits vorgestellten Distanzmaßen überein. Der in dieser Arbeit zur Definition von Signal-Constraints verwendete Gleichheits-Operator unterstützt eine parametrierbare Toleranz t , welche entsprechend berücksichtigt werden muss.

Die Bemessung der Distanz zur Erfüllung bzw. Verletzung eines Signal-Constraints φ ist für ein beispielhaftes Individuum i (bzw. die beiden beispielhaften reellwertigen Signalverläufe $r(\xi_1, i)$ und $r(\xi_2, i)$) für die beiden Vergleichsoperatoren *Größer* ($\varphi \triangleq (\xi_1 > \xi_2)$) und *Gleich* ($\varphi \triangleq (\xi_1 ==_t \xi_2)$) in Abb. 34 illustriert. Abb. 34a geht zunächst auf den den Operator $>$ ein. Das durch die Auswertung des Operators $>$ erzeugte Binärsignal enthält in jedem Datenpunkt d den Wert 1, für den die Größer-Bedingung erfüllt ist (Grundlage für die Berechnung der Constraint-Verletzungsrate v_R). Die entsprechenden Distanzen $\tilde{d}^e$ und $\tilde{d}^v$ ergeben sich jeweils aus der Distanz der Datenwerte der beiden reellwertigen Signale. Für die Operatoren $<$, $\geq$ und $\leq$ ergibt sich eine ähnliche Auswertung. Abb. 34b auf der nächsten Seite illustriert zusätzlich die Berechnungen für den Operator $==_t$. Das obere Diagramm visualisiert den sich aus dem Parameter t

Beispiel

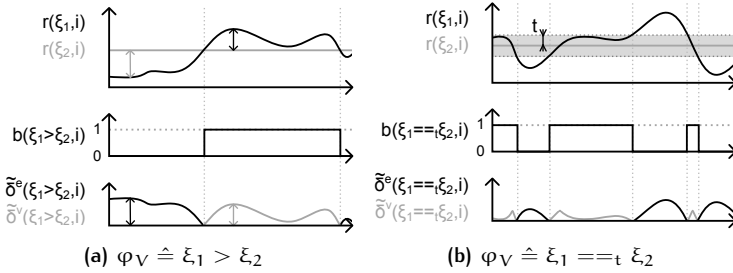


Abb. 34: Beispielhafte Berechnung der Distanzmaße für die Vergleichsoperatoren (a) *Größer* und (b) *Gleich*. Das jeweils obere Diagramm zeigt zwei von den Operatoren ξ_1 und ξ_2 erzeugte reellwertige Signale. Das jeweils mittlere Diagramm veranschaulicht die Constraint-Auswertung $b(\varphi, i)$, während das jeweils untere Diagramm die Distanzmaße $\tilde{\delta}^e$ bzw. $\tilde{\delta}^v$ illustriert.

ergebenen Schlauch um den Signalverlauf des konstanten Signals, welcher während der Auswertung des Constraints berücksichtigt wird.

$\tilde{\delta}^e$ und $\tilde{\delta}^v$ für aussagenlogische Operatoren φ_A

Die Berechnung der Distanzmaße $\tilde{\delta}^e$ und $\tilde{\delta}^v$ basiert bei aussagenlogischen Operatoren φ_A ebenso wie bei den Vergleichsoperatoren φ_V auf dem Berechnen von Distanzen zwischen den einzelnen Datenpunkten der von den untergeordneten Operatoren erzeugten Signale. Während die untergeordneten Operatoren der Vergleichsoperatoren reellwertige Signale erzeugen, erzeugen jene der aussagenlogischen Operatoren Binärsignale. Da durch Binärsignale jedoch keinerlei Distanzaussagen getroffen werden können, werden bei aussagenlogischen Operatoren die Distanzen basierend auf den Distanzmaßen δ^e und δ^d der untergeordneten Operatoren berechnet. Die Berechnung basiert auf der zugrundeliegenden Logik des jeweiligen aussagenlogischen Operators. Tab. 4 auf der nächsten Seite beschreibt die Berechnungen aller unterstützten aussagenlogischen Operatoren. Diese verwenden die vom suchbasierten Strukturtest bekannten Distanzmaße für logische Verknüpfungen ($\delta_{\&}$ und $\delta_{||}$) (vgl. Tab. 1 auf Seite 37).

Beispiel

Abb. 35 illustriert diese Berechnungen für vier aussagenlogische Operatoren. Die Distanz zur Erfüllung des Konjunktionsoperators berechnet sich für jeden Datenpunkt d gemäß $\delta_{\&}$ durch die

φ_A	$\tilde{\delta}_d^e(\varphi_A, i)$	$\tilde{\delta}_d^v(\varphi_A, i)$
$\varphi_1 \wedge \varphi_2$	$\delta_{\&}(\varphi_1^e, \varphi_1^e)$	$\delta_{ }(\varphi_1^v, \varphi_2^v)$
$\varphi_1 \vee \varphi_2$	$\delta_{ }(\varphi_1^e, \varphi_2^e)$	$\delta_{\&}(\varphi_1^v, \varphi_2^v)$
$\varphi_1 \rightarrow \varphi_2$	$\delta_{ }(\varphi_1^v, \varphi_2^e)$	$\delta_{\&}(\varphi_1^e, \varphi_2^v)$
$\varphi_1 \leftrightarrow \varphi_2$	$\delta_{\&}(\delta_{ }(\varphi_1^v, \varphi_2^e),$ $\delta_{ }(\varphi_1^e, \varphi_2^v))$	$\delta_{ }(\delta_{\&}(\varphi_1^e, \varphi_2^v),$ $\delta_{\&}(\varphi_1^v, \varphi_2^e))$
$\varphi_1 \dot{\vee} \varphi_2$	$\tilde{\delta}_d^v(\varphi_1 \leftrightarrow \varphi_2, i)$	$\tilde{\delta}_d^e(\varphi_1 \leftrightarrow \varphi_2, i)$
$\neg \varphi_1$	φ_1^v	φ_1^e
es gilt $\varphi_n^e \triangleq \delta_d^e(\varphi_n, i)$, $\varphi_n^d \triangleq \delta_d^e(\varphi_n, i)$ mit $n \in \{1, 2\}$		

Tab. 4: Berechnung der Distanzmaße $\tilde{\delta}^e$ und $\tilde{\delta}^v$ für die aussagenlogischen Operatoren $\varphi_A \in O_B$ und ein gegebenes Individuum i .

Addition der Signalwerte $\delta_d^e(\varphi_1, i)$ und $\delta_d^e(\varphi_2, i)$. Die Distanz zur Verletzung hingegen berechnet sich gemäß $\delta_{||}$ als der Quotient aus dem Produkt und der Summe der Datenwerte $\delta_d^v(\varphi_1, i)$ und $\delta_d^v(\varphi_2, i)$. Das untere Diagramm illustriert diese Berechnung analog für den Disjunktionsoperator. Abb. 35b illustriert darüber hinaus die Distanzberechnungen für den Implikations- und Äquivalenzoperator. Die Operatoren der Kontravalenz und Negation sind nicht gesondert illustriert. Die Kontravalenz entspricht illustrativ der negierten Äquivalenz und die Negation vertauscht lediglich die Signale $\tilde{\delta}^e$ und $\tilde{\delta}^v$.

$\tilde{\delta}^e$ und $\tilde{\delta}^v$ für temporale Operatoren φ_T

Es wird im Folgenden nur auf jene Varianten der temporalen Operatoren mit beidseitig beschränktem Intervall eingegangen, da sich die restlichen Intervalle auf beidseitig beschränkte Intervalle zurückführen lassen: $[\geq a] \triangleq [a, D - 1]$, $[\leq b] \triangleq [0, b]$. Die Distanzen zur Erfüllung bzw. zur Verletzung für die temporalen Operatoren sind in Tab. 5 und Tab. 6 zusammengefasst und werden im Folgenden informell hergeleitet.

Der *Always* Operator ($\Box$) verlangt die ununterbrochene Gültigkeit der durch den untergeordneten Operator φ definierten Signaleigenschaft innerhalb eines bestimmten Intervalls. Für die schwache Variante des *Always* Operators ($\Box^w \varphi$) mit dem beidseitig be-

Always

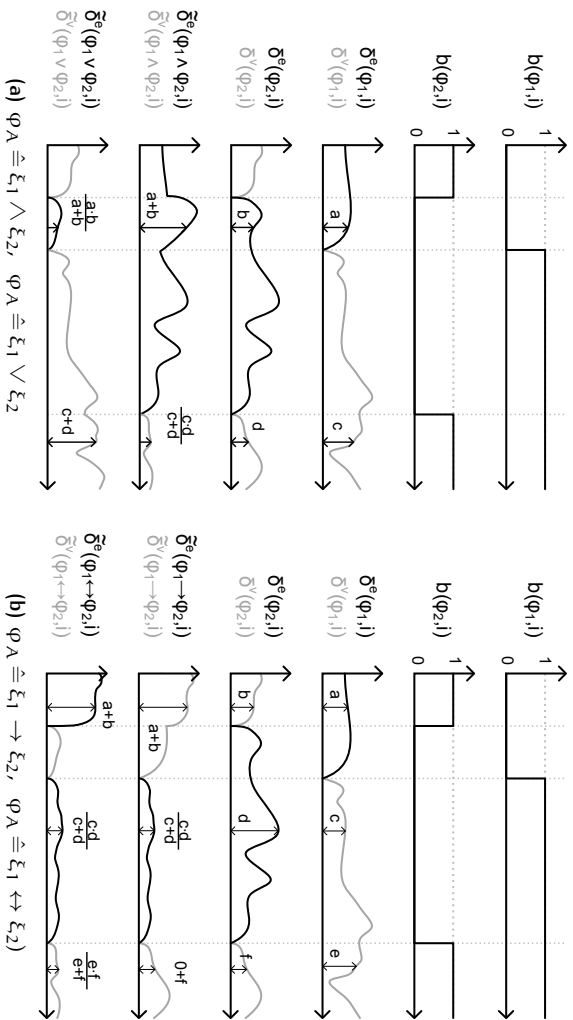


Abb. 35: Beispielhafte Berechnung der Distanzmaße für die aussagenlogischen Operatoren (a) *Konjunktion* und *Disjunktion* bzw. (b) *Implikation* und *Äquivalenz*. Die jeweils oberen beiden Diagramme zeigen zwei von den Operatoren φ_1 und φ_2 erzeugte Binärsignale. Die beiden Diagramme darunter zeigen jeweils beispielhafte Distanzmaße δ^e und δ^v der Operatoren φ_1 und φ_2 . Die jeweils unteren Diagramme illustrieren die Distanzmaße $\tilde{\delta}^e$ bzw. $\tilde{\delta}^v$ für die untersuchten aussagenlogischen Operatoren.

φ_T	$\tilde{\delta}_d^e(\varphi_T, i)$
$\square_{[a,b]}^w \varphi$	$\sum_{t=a'}^{\bar{b}} \delta_t^e(\varphi, i)$
$\square_{[a,b]}^s \varphi$	$\begin{cases} \tilde{\delta}_d^e(\square_{[a,b]}^w \varphi, i) & , \text{ falls } b' < D \\ 0 & , \text{ sonst} \end{cases}$
$\diamond_{[a,b]}^w \varphi$	$\min_{t \in \mathbb{N}_{[a', \bar{b}]}} \delta_t^e(\varphi, i)$
$\diamond_{[a,b]}^s \varphi$	$\begin{cases} \tilde{\delta}_d^e(\diamond_{[a,b]}^w \varphi, i) & , \text{ falls } b' < D \\ 0 & , \text{ sonst} \end{cases}$
$\varphi_1 \mathcal{U}_{[a,b]}^w \varphi_2$	$\delta_{ }(\tilde{\delta}_d^e(\varphi_1 \mathcal{U}_{[a,b]}^s \varphi_2, i), \tilde{\delta}_d^e(\square_{[a,b]}^w \varphi_1, i))$
$\varphi_1 \mathcal{U}_{[a,b]}^s \varphi_2$	$\begin{cases} \sum_{t=a'}^{u-1} \delta_{1,t}^e, \text{ falls } \begin{matrix} \exists u \in \mathbb{N}_{[a', \bar{b}]: \\ b_u(\varphi_2, i) \end{matrix} \\ -b_t(\varphi_1, i) \\ \min_{t \in \mathbb{N}_{(a', \bar{b}]}} \left[\delta_{2,t}^e + \sum_{\substack{u=a' \\ -b_u(\varphi_1, i)}}^{t-1} \delta_{1,u}^e \right], \text{ sonst} \end{cases}$

es gilt $a' \triangleq d + \lceil a/R \rceil$, $b' \triangleq d + \lceil b/R \rceil$, $\bar{b} \triangleq \min(b', D - 1)$,
sowie $\delta_{n,d}^a \triangleq \delta_d^a(\varphi_n, i)$, mit $a \in \{e, v\}$, $n \in \{1, 2\}$, $d \in \mathbb{N}$

Tab. 5: Berechnung der Distanzmaße $\tilde{\delta}^e$ für die wichtigsten temporalen Operatoren $\varphi_T \in \mathcal{O}_B$ und ein gegebenes Individuum i .

schränkten Intervall $[a, b]$ wird für jeden Datenpunkt d verlangt, dass die Datenwerte des Binärsignals von φ für alle Datenpunkte im Intervall $[a', b']$ den Wert 1 aufweisen. Dieses Intervall ist durch eine Zuordnung der Zeitangaben a und b zu den jeweiligen Datenpunkten definiert: $a' = d + \lceil a/R \rceil$ und $b' = d + \lceil b/R \rceil$. Die Distanz zur Erfüllung δ^e lässt sich berechnen als die Summe aller Distanzen zur Erfüllung von φ im untersuchten Intervall. Für eine Verletzung des *Always* Operators genügt es, wenn ein einziger Datenpunkt des Binärsignals von φ im betrachteten Intervall den Wert 0 enthält. Die Distanz zur Verletzung berechnet sich folglich als die im untersuchten Intervall minimale Distanz zur Verletzung von φ . Für die strikte Variante des *Always* Operators ($\square^s \varphi$) muss berücksichtigt werden, ob das untersuchte Intervall innerhalb der

φ_T	$\tilde{\delta}_d^v(\varphi_T, i)$
$\Box_{[a,b]}^w \varphi$	$\min_{t \in \mathbb{N}_{[a', \bar{b}]}} \delta_t^v(\varphi, i)$
$\Box_{[a,b]}^s \varphi$	$\tilde{\delta}_d^v(\Box_{[a,b]}^w \varphi, i)$
$\Diamond_{[a,b]}^w \varphi$	$\sum_{t=a'}^{\bar{b}} \delta_t^v(\varphi, i)$
$\Diamond_{[a,b]}^s \varphi$	$\tilde{\delta}_d^v(\Diamond_{[a,b]}^w \varphi, i)$
$\varphi_1 \mathcal{U}_{[a,b]}^w \varphi_2$	$\begin{cases} \tilde{\delta}_d^v(\varphi_1 \mathcal{U}_{[a,b]}^s \varphi_2, i) & , \text{ falls } \exists t \in \mathbb{N}_{(a', \bar{b}]}: \\ & b_t(\varphi_2, i) \\ \tilde{\delta}_d^v(\Box_{[a,b]}^w \varphi_1, i) & , \text{ sonst} \end{cases}$
$\varphi_1 \mathcal{U}_{[a,b]}^s \varphi_2$	$\min_{t \in \mathbb{N}_{[a', u]}} \delta_{1,t}^v$, mit $\begin{matrix} u \in \mathbb{N}_{(a', \bar{b})}, b_u(\varphi_2, i), \\ \forall v \in \mathbb{N}_{[a', \bar{b}]} \setminus \{u\}: b_v(\varphi_2, i) \rightarrow v > u \end{matrix}$

es gilt $a' \triangleq d + \lceil a/R \rceil$, $b' \triangleq d + \lceil b/R \rceil$, $\bar{b} \triangleq \min(b', D - 1)$,
sowie $\delta_{n,d}^a \triangleq \delta_d^a(\varphi_n, i)$, mit $a \in \{e, v\}$, $n \in \{1, 2\}$, $d \in \mathbb{N}$

Tab. 6: Berechnung der Distanzmaße $\tilde{\delta}^v$ für die wichtigsten temporalen Operatoren $\varphi_T \in O_B$ und ein gegebenes Individuum i .

gültigen Signallänge liegt, d.h. ob gilt: $a' < b' < D$. Ist dies nicht der Fall, so kann die Signaleigenschaft nicht erfüllt sein und es wird der Wert 0 als Distanz zur Erfüllung zurückgegeben. Die Distanz zur Verletzung berechnet sich wie bei der schwachen Variante.

Eventually

Der *Eventually* Operator ($\Diamond$) fordert die mindestens einmalige Gültigkeit der durch den untergeordneten Operator φ definierten Signaleigenschaft innerhalb eines bestimmten Intervalls. Für jeden Datenpunkt d wird verlangt, dass mindestens ein Datenwert des Binärsignals von φ innerhalb des Intervalls $[a', b']$ den Wert 1 aufweist. Liegt dieses Intervall mindestens teilweise außerhalb der gültigen Signallänge, ist die schwache Variante ($\Diamond^w \varphi$) gültig. Die strikte Variante ($\Diamond^s \varphi$) kann demgegenüber nur für jene Datenpunkte gelten, für die das Intervall innerhalb der gültigen Signallänge liegt. Die Distanz zur Erfüllung eines jeden Datenpunkts lässt sich für die schwache Variante des Operators als der minimale Wert aller Distanzen zur Erfüllung von φ innerhalb des Intervalls $[a', b']$ berechnen. Für die strikte Variante muss ähnlich zum *Always* Operator für alle Datenpunkte, für die das

betrachtete Intervall außerhalb der gültigen Signallänge liegt, der Wert 0 zurückgegeben werden. Für die Verletzung des *Eventually* Operators müssen alle Datenpunkte des Binärsignals von φ im jeweils betrachteten Intervall den Wert 0 enthalten. Die Distanz zur Verletzung berechnet sich für die schwache wie auch die strikte Variante als Summe der Distanzen zur Verletzung von φ .

Der *Until* Operator ($\mathcal{U}$) basiert im Gegensatz zu den beiden zuvor beschriebenen temporalen Operatoren auf zwei untergeordneten Operatoren: φ_1 und φ_2 . Dessen strikte Variante ($\mathcal{U}^s$) fordert, dass das Binärsignal von φ_1 im betrachteten Intervall mindestens solange ununterbrochen gültig ist, bis das Binärsignal von φ_2 zum ersten Mal gültig wird. Die schwache Variante ($\mathcal{U}^w$) fordert lediglich die ununterbrochene Gültigkeit des Binärsignals von φ_1 im gesamten betrachteten Intervall, unabhängig von φ_2 . Zur Berechnung der Distanz zur Erfüllung für einen Datenpunkt d muss bei der strikten Variante zunächst überprüft werden, ob ein Datenpunkt $u \in \mathbb{N}_{[a', b']}$ im betrachteten Intervall existiert, an dem das Binärsignal von φ_2 gültig ist. Ist dies der Fall, lässt sich die Distanz zur Erfüllung als die Summe der Distanzen zur Erfüllung von φ_1 im Intervall $[a', u]$ berechnen. Ist dies nicht der Fall, wird für jeden Datenpunkt d im betrachteten Intervall die Summe aus der Distanz zur Erfüllung von φ_2 und der Summe der Distanzen zur Erfüllung von φ_1 bei allen im Intervall vorherigen Datenwerten berechnet. Das Minimum der berechneten Summen ergibt die Distanz zur Erfüllung des strikten *Until* Operators. Ist die Eigenschaft des Operators jedoch erfüllt, so existiert ein gültiger Datenwert u im Binärsignal von φ_2 innerhalb des betrachteten Intervalls $[a', b']$ und alle vorherigen Datenwerte des Binärsignals von φ_1 innerhalb von $[a', u]$ sind ebenfalls gültig. Um den Operator ungültig werden zu lassen, reicht es aus, wenn ein Datenwert des Binärsignals von φ_1 innerhalb von $[a', u]$ auf falsch geändert wird. Die Distanz zur Verletzung berechnet sich demnach als der minimale Wert aller Distanzen zur Verletzung von φ_1 im Intervall $[a', u]$. Für die schwache Variante des *Until* Operators muss der Spezialfall betrachtet werden, dass kein Datenwert von φ_2 innerhalb von $[a', b']$ gültig ist. In diesem Fall berechnet sich die Distanz zur Verletzung wie die Distanz zur Verletzung des *Always* Operators.

Until

Abb. 36 illustriert die Distanzberechnungen für die drei temporalen Operatoren anhand beispielhafter Signalverläufe. Abb. 36a geht zunächst auf die Operatoren *Always* und *Eventually* ein. Für

Beispiel

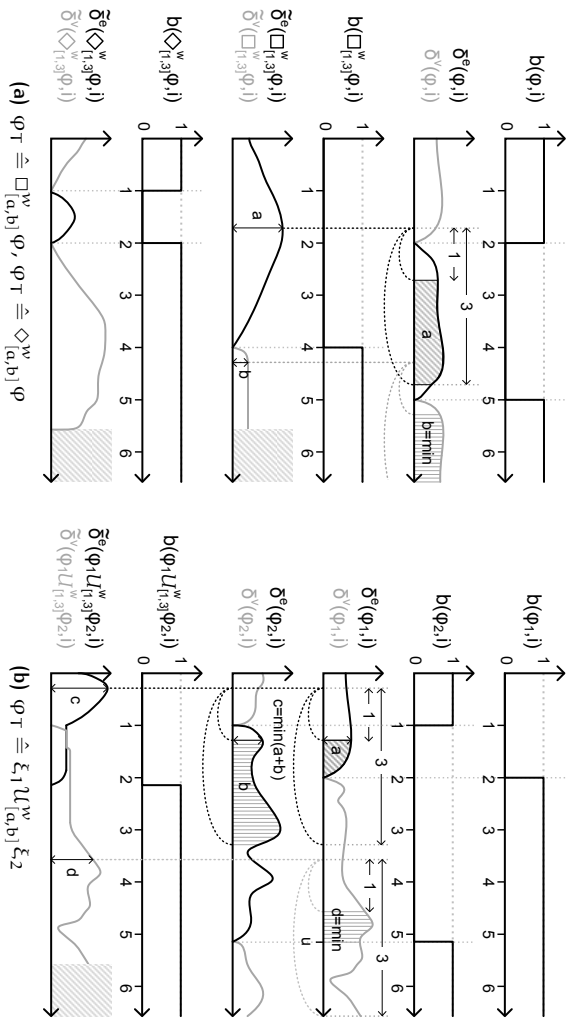


Abb. 36: Beispielhafte Berechnung der Distanzmaße für die temporalen Operatoren (a) *Always* und *Eventually* bzw. (b) *Until*. Die jeweils oberen Diagramme zeigen die von den Operatoren φ bzw. φ_1 und φ_2 erzeugten Binärsignale. Darunter sind jeweils beispielhafte Distanzmaße δ^e und δ^v dieser Operatoren aufgeführt. Die jeweils unteren Diagramme illustrieren die Distanzmaße $\tilde{\delta}^e$ bzw. $\tilde{\delta}^v$ für die untersuchten temporalen Operatoren.

die beiden temporalen Operatoren $\Box^w$ und $\Diamond^w$ mit der Beschränkung auf das Intervall $[a, b] = [1, 3]$ ist jeweils das Binärsignal ihrer Auswertung und die gemäß Tab. 5 bzw. Tab. 6 berechneten Distanzen illustriert. Die Distanz zur Erfüllung des *Always* Operators berechnet sich für den Zeitschritt 1.7 als das Integral der Distanz zur Erfüllung von φ , d.h. als die Summe aller Datenwerte, im Intervall $[a', b'] = [2.7, 4.7]$. Die Distanz zur Verletzung berechnet sich demgegenüber (dargestellt für Zeitschritt 4.7) als der minimale Datenwert aller Distanzen zur Verletzung von φ im Intervall $[a', b'] = [5.3, D - 1]$. Die Distanzen für den *Eventually* Operator berechnen sich analog. Abb. 36b geht anschließend auf den *Until* Operator ein. Für den Zeitschritt 0.4 ist dargestellt, dass sich die Distanz zur Erfüllung aus dem Integral der Distanzen zur Erfüllung von φ_1 (a) und der Distanz zur Erfüllung von φ_2 (b) berechnet. Die Distanz zur Verletzung wird, wie für Zeitschritt 3.6 dargestellt, als das Integral der Distanzen zur Erfüllung von φ_1 im Intervall $[4.6, u]$ berechnet, wobei u jener Zeitschritt ist, an dem das Binärsignal von φ_2 erstmalig gültig wurde.

$\tilde{\delta}^e$ und $\tilde{\delta}^v$ für Ereignis-Operatoren φ_E

Die Ereignis-Operatoren treffen Aussagen über Signaländerungen, wie das Umschalten eines Binärsignals bzw. das Ansteigen oder Fallen eines reellwertigen Signals. Die Distanzen zur Erfüllung bzw. zur Verletzung für die Ereignis-Operatoren sind in Tab. 7 zusammengefasst und werden im Folgenden informell hergeleitet.

Die beiden in der STL von Maler und Nickovic enthaltenen Ereignis-Operatoren *Rise* ($\downarrow \varphi$) und *Fall* ($\downarrow \varphi$) basieren auf Binärsignalen, welche vom untergeordneten Operator φ erzeugt werden. Sie werden zu wahr ausgewertet, wann immer das Binärsignal von φ von 0 zu 1 bzw. von 1 zu 0 wechselt. Die Distanz zur Erfüllung lässt sich für diese Operatoren sehr einfach durch Verwendung der Distanzberechnung für Konjunktionen ($\delta_\&$) berechnen: Sie berechnet sich für Zeitschritt d als die Distanz zur Verletzung von φ im Zeitschritt zuvor ($d - 1$) konjugiert mit der Distanz zur Erfüllung von φ in Zeitschritt d. Für die Gültigkeit des *Fall* Operators in Zeitschritt d wird analog die Konjunktion der Distanz zur Erfüllung in Zeitschritt $d - 1$ mit der Distanz zur Verletzung in Zeitschritt d berechnet. Für die Berechnung der Distanz zur Verletzung im Falle des *Rise* Operators wird die Distanz zur Erfüllung von φ in Zeitschritt $d - 1$ disjunktiv mit

Rise & Fall

φ_E	$\tilde{\delta}_d^e(\varphi_E, i)$	$\tilde{\delta}_d^v(\varphi_E, i)$
$\uparrow \varphi$	$\delta_{\&}(\delta_{d-1}^v, \delta_d^e)$	$\delta_{ }(\delta_{d-1}^e, \delta_d^v)$
$\downarrow \varphi$	$\delta_{\&}(\delta_{d-1}^e, \delta_d^v)$	$\delta_{ }(\delta_{d-1}^v, \delta_d^e)$
$\uparrow_{[\geq a]} \xi$	$\delta_{>}(r_d(\xi, i), r_{a'}(\xi, i))$	$\delta_{\leq}(r_d(\xi, i), r_{a'}(\xi, i))$
$\downarrow_{[\geq a]} \xi$	$\delta_{<}(r_d(\xi, i), r_{a'}(\xi, i))$	$\delta_{\geq}(r_d(\xi, i), r_{a'}(\xi, i))$
$\uparrow_{[\geq a]} \xi$	$\delta_{\geq}(c_{\uparrow}, a)$	$\delta_{<}(c_{\uparrow}, a)$
$\downarrow_{[\geq a]} \xi$	$\delta_{\geq}(c_{\downarrow}, a)$	$\delta_{<}(c_{\downarrow}, a)$
$\uparrow_{[a,b]} \xi$	$\delta_{\&}(\delta_{\geq}(c_{\uparrow}, a), \delta_{\leq}(c_{\uparrow}, b))$	$\delta_{ }(\delta_{<}(c_{\uparrow}, a), \delta_{>}(c_{\uparrow}, b))$
$\downarrow_{[a,b]} \xi$	$\delta_{\&}(\delta_{\geq}(c_{\downarrow}, a), \delta_{\leq}(c_{\downarrow}, b))$	$\delta_{ }(\delta_{<}(c_{\downarrow}, a), \delta_{>}(c_{\downarrow}, b))$

es gilt $\delta_d^a \triangleq \delta_d^a(\varphi, i)$, mit $a \in \{e, v\}$, $d \in \mathbb{N}$

$$c_{\uparrow} = \left| \left\{ t \in \mathbb{N}_{(d,D)} \mid \forall u \in \mathbb{N}_{(d,t)} : r_u(\xi, i) \geq r_{u-1}(\xi, i) \right\} \right|$$

$$c_{\downarrow} = \left| \left\{ t \in \mathbb{N}_{(d,D)} \mid \forall u \in \mathbb{N}_{(d,t)} : r_u(\xi, i) \leq r_{u-1}(\xi, i) \right\} \right|$$

Tab. 7: Berechnung der Distanzmaße $\tilde{\delta}^e$ und $\tilde{\delta}^v$ für die Ereignis-Operatoren $\varphi_E \in O_B$ und ein gegebenes Individuum i .

der Distanz zur Verletzung in Zeitschritt d verknüpft. Die Distanz zur Verletzung für den *Fall* Operator berechnet sich analog.

Die folgenden Ereignis-Operatoren basieren auf jeweils einem reellwertigen Signal. Sie drücken aus, ob dieses Signal innerhalb eines bestimmten Intervalls steigt oder fällt. Die *Rise* und *Fall* Operatoren in ihrer schwachen Form ($\uparrow_{[\geq a]}$ und $\downarrow_{[\geq a]}$) fordern für jeden Datenpunkt d , dass das reellwertige Signal des jeweils untergeordneten Operators ξ innerhalb des Intervalls $[d, a']$ steigt bzw. fällt. Es wird lediglich geprüft, ob der Signalwert in Zeitschritt a' größer bzw. kleiner ist als jener in Zeitschritt d . Sowohl die Distanz zur Erfüllung als auch die Distanz zur Verletzung lässt sich für diese Operatoren sehr einfach mit Hilfe der Distanzmaße $\delta_{\circ}$, mit $\circ \in \{>, <, \geq, \leq\}$ berechnen (vgl. Tab. 1 auf Seite 37).

Die strikten Varianten der *Rise* und *Fall* Operatoren fordern zusätzlich, dass der Signalanstieg bzw. -abfall monoton ist. Zur Berechnung der Distanzen zur Erfüllung bzw. zur Verletzung wird der Parameter $c_{\uparrow}$ bzw. $c_{\downarrow}$ verwendet. Diese Parameter be-

stimmen, wie viele Zeitschritt das zugrundeliegende Signal seit dem Zeitschritt d monoton ansteigt bzw. abfällt. Die Distanzen zur Erfüllung für die strikten *Rise* und *Fall* Operatoren lassen sich dann mit Hilfe des Größer-Gleich-Distanzmaßes ($\delta_{\geq}$) für die Bedingungen $c_{\uparrow} \geq a$ bzw. $c_{\downarrow} \geq a$ berechnen. Analog lassen sich die Distanzen zur Verletzung mit Hilfe des Kleiner-Distanzmaßes ($\delta_{<}$) für die Bedingungen $c_{\uparrow} < a$ bzw. $c_{\downarrow} < a$ berechnen.

Die strikten *Rise* und *Fall* Operatoren mit beidseitig beschränktem Intervall fordern zusätzlich, dass der monotone Anstieg bzw. Abfall des Signals sowohl eine minimale als auch eine maximale Anzahl von Zeitschritten nicht überschreitet. Die Anzahl der Zeitschritte, in denen das untersuchte Signal steigt ($c_{\uparrow}$) bzw. fällt ($c_{\downarrow}$), muss innerhalb der definierten Intervallgrenzen (a und b) liegen. Es kann dazu wieder Gebrauch von den Distanzmaßen für die Konjunktion bzw. die Disjunktion gemacht werden (vgl. Tab. 1 auf Seite 37). Das selbe gilt für die Berechnung der Distanzen zur Verletzung.

Abb. 37 und Abb. 38 illustrieren die Distanzberechnungen für die Ereignis-Operatoren anhand beispielhafter Signalverläufe. Abb. 37a geht zunächst auf den *Rise* Operator für Binärsignale ein. Es ist zu sehen, dass das resultierende Binärsignal der Auswertung des *Rise* Operators für jeden einzelnen Datenpunkt genau dann den Wert 1 besitzt, wenn $b(\varphi, i)$ auf den Wert 1 wechselt. Für die Berechnung der Distanzen δ^e und δ^v in Datenpunkt d werden die Zeitschritte d und $d - 1$ berücksichtigt. Die Distanz zur Erfüllung in Zeitschritt 5 berechnet sich also beispielsweise als die Summe aus 0 ($\delta_5^v(\varphi, i)$) und a ($\delta_4^e(\varphi, i)$), jene in Zeitschritt 8 aus der Summe aus b ($\delta_7^e(\varphi, i)$) und c ($\delta_8^v(\varphi, i)$). Abb. 37b illustriert die Berechnungen für den schwachen *Rise* Operator mit einer geforderten minimalen Anzahl von fünf Zeitschritten. Für die Berechnung von δ^e und δ^v wird für jeden Datenpunkt d der Datenpunkt $d + 5$ herangezogen. Die Distanz zur Erfüllung berechnet sich beispielsweise für Zeitschritt 11 als die Differenz des Minuenden $c(r_{11}(\xi, i))$ und des Subtrahenden $a(r_{16}(\xi, i))$. Für Datenpunkt 16 berechnet sich die Distanz zur Verletzung als die Differenz aus $c(r_{16}(\xi, i))$ und $d(r_{21}(\xi, i))$. Abb. 38a illustriert die Distanzberechnungen für den einseitig beschränkten *Rise* Operator ($\uparrow_{[\geq 5]}$). Das resultierende Binärsignal ist in diesem Beispiel lediglich in Zeitschritt 11 erfüllt, da nur von diesem Zeitschritt aus $r(\xi, i)$ mindestens 5 Zeitschritte lang monoton steigt. Die Distanz zur Verletzung berechnet sich in diesem Fall gemäß $\delta_{<}$

Beispiel

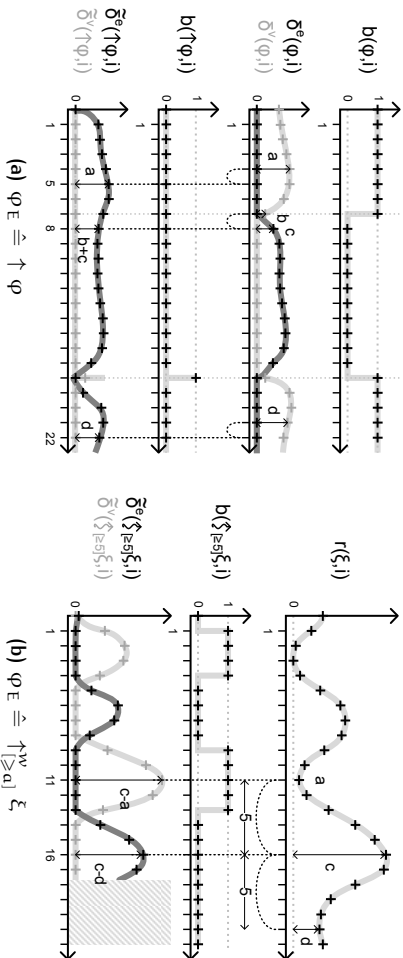


Abb. 37: Beispielhafte Berechnung der Distanzmaße für die Ereignis-Operatoren *Rise* (a) für Binärsignale und (b) für reellwertige Signale. Die jeweils obersten Diagramme zeigen die von den Operatoren φ bzw. ξ erzeugten Signale. Im ersten Fall sind darunter die zugrundeliegenden Distanzwerte angegeben. Folgend sind jeweils die durch Auswertung der Operatoren erzeugten Binärsignale dargestellt. Das jeweils unterste Diagramm illustriert die berechneten Distanzen δ^e und δ^v .

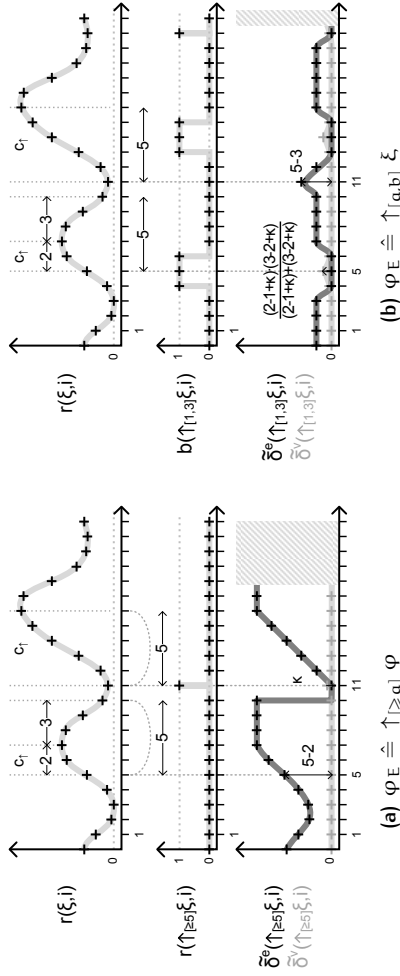


Abb. 38: Beispielhafte Berechnung der Distanzmaße für die Ereignis-Operatoren *Rise* (a) mit einseitig und (b) zweiseitig beschränktem Intervall. Die jeweils obersten Diagramme zeigen die von den Operatoren ξ erzeugten reellwertigen Signale. Folgend sind jeweils die durch Auswertung der Operatoren erzeugten Binärsignale dargestellt. Das jeweils unterste Diagramm illustriert die berechneten Distanzen $\tilde{\delta}^e$ und $\tilde{\delta}^v$ für die untersuchten temporalen Operatoren.

als $c_{\uparrow} - a + \kappa \Rightarrow 5 - 5 + \kappa = \kappa$. Für die restlichen Datenpunkte, wie beispielsweise in Zeitschritt 5, berechnet sich die Distanz zur Erfüllung als die Differenz aus a und $c_{\uparrow}$. Abb. 38b veranschaulicht abschließend die Distanzberechnungen für den beidseitig beschränkten *Rise* Operator. Die Distanzen δ^e und δ^v berechnen sich jeweils gemäß Tab. 7 unter Verwendung der Distanzmaße $\delta_{\&}$ bzw. $\delta_{||}$.

3.4 ZUSAMMENFASSUNG

Dieses Kapitel befasste sich mit der Generierung und Optimierung von Signalen im Rahmen des suchbasierten Modelltests.

*Ansätze zur
Signal-
generierung*

Zu Beginn wurde in Abschnitt 3.1 auf die prinzipielle Herangehensweise und die notwendige Trennung zwischen genotypischer und phänotypischer Repräsentation eingegangen. Darauf aufbauend wurden drei Signalgenerierungsverfahren vorgestellt: Eines verwendet ein parametrierbares trigonometrisches Polynom, während die beiden anderen Verfahren auf parametrierbaren Signalsegmenten basieren. Sowohl die Parameter des Polynoms als auch die Parameter der einzelnen Signalsegmente definieren den genotypischen Suchraum, auf welchem das zur Anwendung kommende Such- bzw. Optimierungsverfahren operiert. Lösungsvorschläge aus dem genotypischen Suchraum werden durch einen entsprechenden Dekoder (Signalgenerator) in Lösungsvorschläge im phänotypischen Suchraum transformiert, d.h. aus den optimierten Parametern werden ausführbare Signale erstellt. Die vorgestellten Signalgenerierungsverfahren wurden zusätzlich im Rahmen einer experimentellen Fallstudie vergleichend untersucht.

Bewertung

Die im Rahmen des suchbasierten Tests notwendige Bewertung der erstellten phänotypischen Lösungsvorschläge wurde in Abschnitt 3.2 beschrieben. Die Besonderheit der suchbasierten Testdatengenerierung für zustandsbehaftete Systeme besteht darin, dass die generierten Teststimuli aus einer Vielzahl von Zeitschritten bestehen können, die separat bewertet werden müssen. Die verwendeten Optimierungsverfahren erwarten jedoch einen einzelnen Zielfunktionswert als Bewertung eines jeden Lösungsvorschlags. Aus diesem Grund müssen die Bewertungen aller Zeitschritte eines jeden Lösungsvorschlags zu einer Gesamtbewertung zusammengefasst werden.

Eine weitere Notwendigkeit bei der Generierung relevanter Teststimuli ist die Berücksichtigung potentiell vorhandener komplexer Constraints und Abhängigkeiten der Testobjekteingaben. In Abschnitt 3.3 wurde daher ein umfangreiches Konzept zur Spezifikation und Auswertung von Signal-Constraints entwickelt. Die Constraints werden mit Hilfe einer speziellen temporalen Logik spezifiziert und können für gegebene Signale evaluiert und ausgewertet werden. Für jedes Signal kann dadurch eine feine Bewertung der Erfüllung bzw. Nicht-Erfüllung der Constraints berechnet werden. Diese Bewertung wird unter Anwendung einer Penalty Funktion in den Testdatengenerierungs- bzw. Optimierungsprozess einbezogen.

*Signal-
Constraints*

4

ÜBERDECKUNG VON SIMULINK/STATEFLOW

Der Weg ist das Ziel.

— Konfuzius

Zur Anwendung des suchbasierten Strukturtests auf die mit ML/SL/SF modellierten Systeme müssen die generierten Teststimuli bezüglich ihrer Eignung, einzelne strukturelle Elemente (z.B. SL Blöcke oder SF Diagramme) zu überdecken, bewertet werden. Dieses Kapitel geht auf diese Bewertung ein.

*Bewertung
der
Teststimuli*

Zunächst werden in Abschnitt 4.1 die für SL/SF anwendbaren Überdeckungskriterien diskutiert. Die für die einzelnen strukturellen Elemente zur Bewertung der Teststimuli notwendigen Beobachtungen während der Ausführung des Testobjekts werden identifiziert. Die entsprechend notwendige Instrumentierung der SL Blöcke und SF Diagramme wird in Abschnitt 4.2 behandelt. Basierend auf diesen Beobachtungen kann eine Distanz zum Erreichen der Testziele berechnet werden, welcher als Bewertung der Teststimuli an das Optimierungsverfahren zurückgegeben wird. Abschnitt 4.3 befasst sich mit den dazu notwendigen Distanzmetriken. Abschnitt 4.4 fasst das Thema der Überdeckung von SL/SF Modellen zusammen.

Gliederung

4.1 ÜBERDECKUNGSKRITERIEN

In Abschnitt 2.1.5 auf Seite 19 wurde auf unterschiedliche für den Strukturtest angewandte kontrollflussorientierte Überdeckungskriterien eingegangen. Diese lassen sich auch für grafische Programmierungssprachen – wie SL/SF – interpretieren [153]. Die in Abschnitt 2.2.3 auf Seite 42 vorgestellten kommerziellen Werkzeuge beispielsweise verwenden eben diese Kriterien zur Generierung relevanter Teststimuli. Daher dienen sie auch in dieser Arbeit als

Grundlage für die Steuerung der Suche nach adäquaten Teststimuli.

Eigene
Inter-
pretation

Aufgrund der zeitschrittorientierten Semantik von SL, werden zu jedem Zeitschritt stets alle Blöcke eines SL Modells ausgeführt. Ausnahmen bilden lediglich aktivierbare Blöcke, wie beispielsweise der Block *enabled subsystem*. Das kontrollflussorientierte Konzept des Erreichens eines strukturellen Elements, d.h. dessen gezielte Ausführung, ist daher nicht auf SL Modelle anwendbar. Dennoch lassen sich die jeweiligen Überdeckungskriterien für SL/SF interpretieren. So stellt die DC Anforderungen an eine Vielzahl bedingter SL/SF Elemente. Ebenso ist sowohl die CC als auch die MC/DC anwendbar: Diese stellen Anforderungen an die atomaren Bedingungen der Entscheidungen.

DC

In Tab. 8 sind die für die drei Überdeckungskriterien DC, CC und MC/DC relevanten SL/SF Elemente dargestellt. Ein Punkt deutet an, dass das jeweilige Objekt für das angegebene Überdeckungskriterium berücksichtigt wird. Ein Großteil aller aufgeführten SL/SF Elemente ist für die DC relevant. Bei diesen Objekten werden die Ergebnisse von Vergleichen als einzelne Entscheidungen interpretiert, welche es zu überdecken gilt. Der *Abs* Block beispielsweise erhält eine Eingabe x und liefert dessen absoluten Wert als Ausgabe y . Die beiden im Rahmen der DC zu überdeckenden Entscheidungen sind gegeben als die Ergebnisse der beiden Vergleiche: $x < 0$ und $x \geq 0$. Die Entscheidungen aller für die CC und MC/DC relevanten Blöcke basieren auf mehreren atomaren Bedingungen, an welche spezifische Anforderungen gestellt werden. Für den *Logical Operator* beispielsweise fordert die CC für alle Eingaben die jeweils mindestens einmalige Auswertung zu *wahr* und zu *falsch*. Zur Erfüllung der MC/DC hingegen muss jede Eingabe mindestens einmal ausschließlich für die Auswertung der Entscheidung jeweils zu *wahr* und zu *falsch* verantwortlich sein.

CC und
MC/DC

Notwendige
Beobach-
tungen

Alle aufgelisteten SL/SF Elemente lassen sich wie in Abb. 39 auf Seite 128 dargestellt abstrahieren. Sie können zeitliche Informationen oder Daten früherer Simulationszeitschritte verwenden (1), besitzen mindestens eine Eingabe (2), können für die Überdeckung relevante interne Parameter besitzen (3) und besitzen mindestens eine Ausgabe (4). Bei SF Diagrammen muss darüber hinaus die Aktivität der enthaltenen Zustände (5) und der enthaltenen Transitionen (6) beobachtet werden. All diese Eigenschaften sind für die Untersuchung der Erfüllung der erwähnten Überdeckungskriterien potentiell relevant. Bei der gesteuerten (suchbasierten)

SL/SF Elemente	DC	CC	MC/DC
Abs	•		
Combinatorial Logic	•	•	
Dead Zone	•		
Discrete-Time-Integrator	•		
Embedded MATLAB Function	•	•	•
Enabled Subsystem	•	•	•
Fcn (bei booleschen Operatoren)		•	
For Iterator	•		
Logical Operator	•	•	•
MinMax	•		
Model	•	•	•
Multiport Switch	•		
Rate Limiter	•		
Relay	•		
Saturation	•		
Stateflow Charts / Transitions	•	•	•
Switch	•		
Switch Case	•		
Triggered Subsystem	•	•	•
While Iterator	•		

Tab. 8: Übersicht über die für die drei Überdeckungskriterien DC, CC und MC/DC relevanten SL/SF Elemente (nach [125]).

Erstellung relevanter Teststimuli, müssen diese Informationen während der Ausführung protokolliert und zur Berechnung von Distanzwerten ausgewertet werden.

Als Beispiel sei hier der *Rate Limiter* aufgeführt: Dieser beschränkt die Steigung eines Signals auf einen minimalen bzw. maximalen Wert. Die CC fordert, dass die Steigung des Eingangssignals beide Schranken jeweils mindestens einmal über- und unterschreitet. Zur Untersuchung, ob die DC erfüllt ist, muss die Steigung des Eingangssignals berechnet und mit der jeweiligen Schranke verglichen werden. Zur Berechnung der Steigung wird der Wert des Eingangssignals, des Ausgangssignals und die Simulationszeit zum aktuellen und zum jeweils vorherigen Simulationszeitschritt benötigt – (1), (2) und (4). Zur Bestimmung einer Distanz zum

Beispiel
Rate Limiter

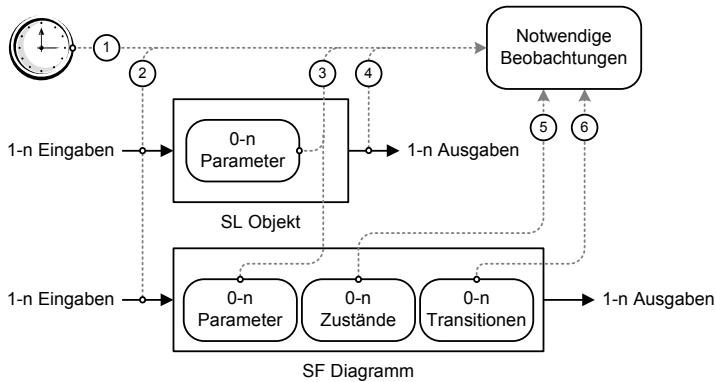


Abb. 39: Zur Berechnung der Annäherung an ein Überdeckungskriterium relevante und zu beobachtende Eigenschaften eines SL Objekts bzw. eines SF Diagramms.

Über- bzw. Unterschreiten einer der Schranken, müssen die als Blockparameter gegebenen Schranken ausgelesen werden (3).

Stateflow

Für SF Diagramme und enthaltene SF Transitionen sind ähnliche Informationen notwendig. Für die DC wird das Auslösen von Transitionen als Entscheidung interpretiert. Die CC und die MC/DC stellen demgegenüber Anforderungen an die einzelnen atomaren Bedingungen, welche zum Auslösen der Transitionen nötig sind. Die zur Beurteilung der Überdeckung notwendigen Beobachtungen umfassen alle in den Bedingungen verwendeten Parameter.

4.2 INSTRUMENTIERUNG

Dieser Abschnitt beschreibt die technische Realisierung der Instrumentierung für SL Blöcke und SF Diagramme. Die durchgeführte Instrumentierung hat das Ziel, die im vorherigen Abschnitt identifizierten Beobachtungen während der Ausführung des Testobjekts zu protokollieren.

Nach der Ausführung steht für jeden Zeitschritt grundsätzlich der Systemtakt sowie alle Datenwerte der protokollierten Signale für die vorherigen und nachfolgenden Zeitschritte zur Verfügung (Beobachtung 1). Ebenso sind die Parameter der SL Blöcke und

SF Diagramme jederzeit auslesbar und stehen somit ebenfalls zur Verfügung (Beobachtung 3).

Die restlichen notwendigen Beobachtungen sind blockspezifisch und werden für SL und SF gesondert betrachtet. Abschnitt 4.2.1 geht zunächst auf die notwendige Instrumentierung für SL Blöcke ein, während sich Abschnitt 4.2.2 anschließend mit der Instrumentierung von SF Diagrammen befasst.

4.2.1 SIMULINK BLÖCKE

In Abschnitt 4.1 wurde das Beobachten der Ein- und Ausgaben (Beobachtungen 2 und 4) der SL Objekte während der Ausführung des Testobjekts mit den generierten Teststimuli als notwendige Voraussetzung für die Bestimmung der Annäherung an die Erfüllung der berücksichtigten Überdeckungskriterien identifiziert.

Voraussetzungen

Jede Eingabe I_n und jede Ausgabe O_n der für die einzelnen Überdeckungskriterien relevanten SL Objekte muss während der Simulation protokolliert werden, sodass sich entsprechend Signale $\log_{I_n}$ und $\log_{O_n}$ ergeben. Dies kann in SL durch Aktivierung der *Simulink Signal Logging* Funktionalität für die jeweiligen Signale realisiert werden. Eine gesonderte Instrumentierung von Ausgabe-Blöcken, wie in der Arbeit von Zhan [153] vorgeschlagen, ist nicht nötig (vgl. Abschnitt 2.2.2 auf Seite 31). Basiert der zu instrumentierende Block auf booleschen Eingaben, wie beispielsweise das *Enabled Subsystem* oder der *Logical Operator*, reicht das alleinige protokollieren ihrer Eingabewerte nicht aus, da eine Distanzberechnung basierend auf booleschen Werten in der Regel nicht zielführend ist. In diesem Fall müssen die Eingaben all jener Blöcke instrumentiert werden, deren Ausgaben die Eingabe für den zu instrumentierenden Block darstellen. Während der Bewertung der Distanzbemessung basierend auf diesen Beobachtungen, muss die Semantik dieser Blöcke entsprechend berücksichtigt werden.

Realisation

Abb. 40 auf der nächsten Seite illustriert diese Instrumentierung beispielhaft. Dargestellt ist ein für alle drei betrachteten Überdeckungskriterien relevanter *Logical Operator*, dessen Eingaben sich aus den Auswertungen zweier Relationsoperatoren ergeben. Da der *Logical Operator* auf booleschen Eingaben basiert, werden stattdessen die Eingaben der beiden Relationsoperatoren $>$ und

Beispiel

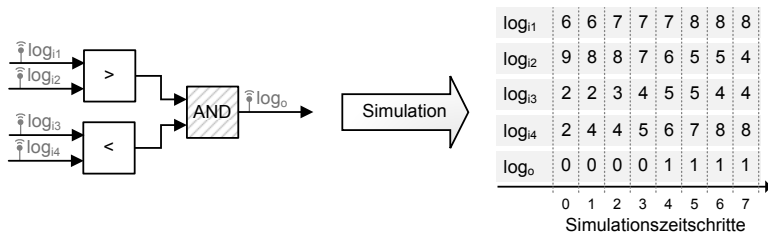


Abb. 40: Instrumentierung von Simulink Blöcken. Besitzt der zu instrumentierende Block boolesche Eingaben, müssen die jeweils vorgelagerten Blöcke instrumentiert werden.

< instrumentiert. Rechts sind protokollierte Signale dargestellt, wie sie sich durch Ausführung des Modells ergeben könnten.

4.2.2 STATEFLOW DIAGRAMME

In Abschnitt 4.1 wurde das Protokollieren der Eingaben eines SF Diagramms, der aktiven Zustände und der Transitionen als notwendige Voraussetzungen für eine Bewertung der Teststimuli bzgl. der Erfüllung eines Überdeckungskriteriums identifiziert. Die folgenden Abschnitte befassen sich mit der dazu notwendigen Instrumentierung der SF Diagramme, bezugnehmend auf das Modell der Automatiktriebsteuerung (vgl. Abschnitt 2.1.2 auf Seite 13).

EINGABEN

Voraussetzung

Alle Eingaben des untersuchten SF Diagramms sind zur Bestimmung der Annäherung an die Erfüllung einer angestrebten Entscheidung oder Bedingung potentiell relevant. Aus diesem Grund wird jede Eingabe I_n für jeden Simulationszeitschritt t in gesonderten Signalen $\log_{I_n}(t)$ protokolliert.

Realisation und Beispiel

Zu diesem Zweck kann ebenfalls auf die *Stateflow Signal Logging* Funktionalität von SF zurückgegriffen werden. Beispielhaft ist dies in Abb. 41a illustriert. Für die drei Eingaben *speed*, *up_th* und *down_th* des SF Diagramms wird die Protokollierung während der Simulation aktiviert – dargestellt durch das graue Antennensymbol. Die sich beispielhaft ergebenden Signale sind rechts

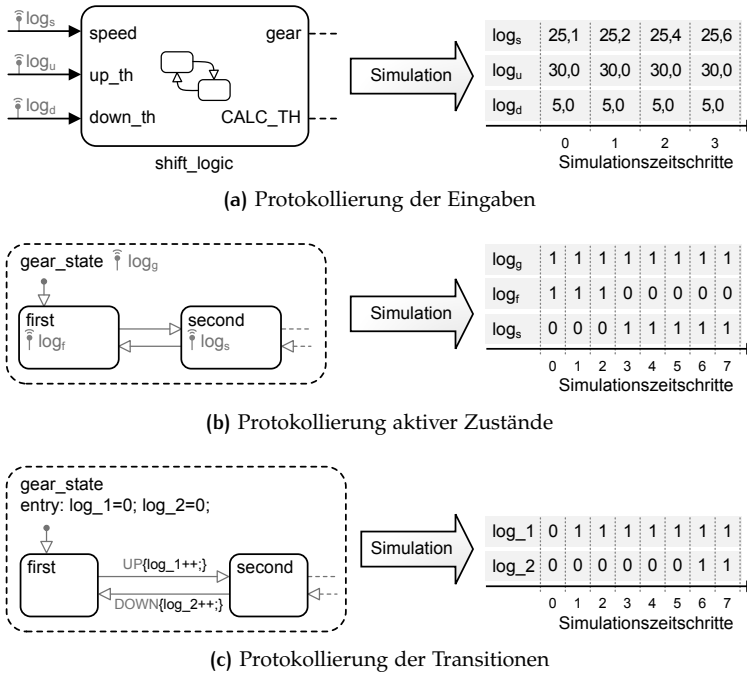


Abb. 41: Instrumentierung von Stateflow Diagrammen.

dargestellt: Es handelt sich um die einzelnen Datenwerte der jeweiligen Eingabe für jeden simulierten Zeitpunkt.

AKTIVE ZUSTÄNDE

In welchem Zustand sich das überwachte SF Diagramm zu jedem Zeitschritt befindet, ist eine besonders wichtige zu überwachende Information. Dies wird durch das Erstellen eines Beobachtungssignals $\log_{S_n}(t)$ für jeden enthaltenen Zustand S_n realisiert, welches die Aktivität des jeweiligen Zustands wie folgt protokolliert:

$$\log_{S_n}(t) = \begin{cases} 1 & \text{Zustand } S_n \text{ ist im Zeitschritt } t \text{ aktiv,} \\ 0 & \text{sonst.} \end{cases} \quad (4.1)$$

Zu diesem Zweck kann die *Stateflow Signal Logging* Funktionalität von SF für jeden zu überwachenden Zustand aktiviert werden.

*Voraus-
setzung*

Realisation

Ein gesonderte Instrumentierung ist nicht notwendig. Nach der Simulation lassen sich die angelegten Signale entsprechend auslesen.

In Abb. 41b ist ein Ausschnitt des SF Diagramms bzw. des Zustands `gear_state` dargestellt. Der Inhalt der instrumentierten Variablen $\log_g$, $\log_f$ und $\log_s$ für die drei dargestellten Zustände nach einer beispielhaften Simulation ist rechts zu sehen. Da es sich bei Zustand `gear_state` um einen parallelen Zustand auf erster Ebene des SF Diagramms handelt, ist dieser zu jedem Zeitschritt aktiv; Die Variable $\log_g$ enthält für jeden Zeitschritt den Wert 1.

TRANSITIONEN

Voraussetzung Zur Bestimmung der Annäherung an eine auszulösende Transition innerhalb des SF Diagramms ist die Rekonstruktion des während der Simulation aufgetretenen Kontrollflusses notwendig. Sobald komplexe Transitionen im SF Diagramm vorhanden sind, d.h. mehrere aufeinander folgende, lediglich durch Verbindungspunkte miteinander verbundene Transitionen, können allein durch die Analyse der protokollierten Signale $\log_s$ keine Rückschlüsse auf den aufgetretenen Kontrollfluss gezogen werden. Aus diesem Grund müssen SF Transitionen ebenfalls instrumentiert werden.

Realisation Dazu muss das SF Diagramm für jede enthaltene Transition T_n mit jeweils einer auf 0.0 initialisierten Variablen $\log_{T_n}$ instrumentiert werden. Diese Variablen werden bei jeder Auslösung der korrespondierenden Transition durch Einbettung als *Transition Action* um 1.0 erhöht. Im Anschluss an die Simulation lässt sich der während der Ausführung aufgetretene Kontrollfluss durch die Analyse rekonstruieren, welche dieser Variablen in welchem Zeitschritt inkrementiert wurde.

Beispiel Abb. 41c illustriert die Instrumentierung zweier Transitionen beispielhaft für einen Ausschnitt der Automatikgetriebesteuerung. Beim Betreten des Zustands `gear_state` werden beide instrumentierten Variablen auf 0 gesetzt und bei Aktivierung der korrespondierenden Transition um 1 erhöht. Eine beispielhafte Protokollierung ist rechts zu sehen: Im ersten Zeitschritt befand sich das System in Zustand `first`. In Zeitschritt 1 ist es in den Zustand `second` gewechselt, um in Zeitschritt 6 wieder in Zustand `first` zu wechseln.

4.3 DISTANZMETRIKEN

Die Instrumentierung liefert spezifische Informationen über die Ausführung des Testobjekts. Für die Bewertung der verwendeten Teststimuli muss die Annäherung an ein bestimmtes Überdeckungsziel berechnet werden. Zu diesem Zweck kommt eine Vielzahl unterschiedlicher Distanzmetriken innerhalb der Zielfunktionsberechnung zum Einsatz. Distanzen zu atomaren Bedingungen ($\delta_=$, $\delta_{\neq}$, $\delta_{\geq}$, $\delta_{\leq}$, $\delta_{<}$, $\delta_{>}$) sowie Distanzen zu logischen Verknüpfungen ($\delta_{\&}$, $\delta_{||}$, δ_{-}) werden berechnet, wie in Abschnitt 2.2.2 auf Seite 31 beschrieben.

Die folgenden Abschnitte stellen vor, wie diese spezifischen Distanzmetriken zur Berechnung der Annäherung an einzelne SL Blöcke und für SF Diagramme verwendet werden können.

4.3.1 SIMULINK BLÖCKE

Die für die einzelnen Überdeckungskriterien relevanten SL Blöcke wurden in Tab. 8 aufgeführt. Diese wurden bezüglich der Voraussetzungen untersucht, welche zur Erfüllung der Überdeckungskriterien DC und CC notwendig sind. Es hat sich gezeigt, dass die notwendige Distanzberechnung in die drei im Folgenden aufgeführten Klassen unterteilt werden kann.

EINGABE MUSS SPEZIFISCHEN WERT AUFWEISEN

Die $n \in \mathbb{N}$ Dateneingaben $I_i, i \in [1, n]$ des SL Blocks *Multiport Switch* werden in Abhängigkeit der Kontrolleingabe I_C als Ausgabe des Blocks O weitergeschaltet. Das mindestens einmalige Weiterschalten einer jeden Dateneingabe wird vom Überdeckungskriterium DC gefordert. An der Kontrolleingabe I_C müssen demnach separat die Werte $1, 2, \dots, n$ anliegen.

*Multiport
Switch*

Der Block *Switch Case* realisiert eine Fallunterscheidung ($n \in \mathbb{N}$ Fälle) durch Aktivierung der assoziierten Subsysteme in Abhängigkeit der Kontrolleingabe I_C . Wie beim Multiport Switch fordert die DC die mindestens einmalige Aktivierung eines jeden Falls. Die Kontrolleingabe I_C muss ebenfalls jeweils die Werte $1, 2, \dots, n$ aufweisen.

Switch Case

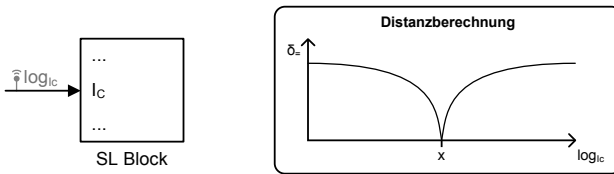


Abb. 42: Distanzberechnung für die Eingabe I_C eines Simulink Blocks zu einem spezifischen Wert x . Die Distanz kann unter Verwendung des Distanzoperators der Identität berechnet werden: $\delta = (\log_{I_C}, x)$.

Distanz- berechnung

In beiden Fällen besteht die Herausforderung darin, eine Eingabe I_C des Blocks auf einen spezifischen Wert zu bringen. Die Distanz zur Erfüllung des gewünschten Testziels lässt sich gemäß $\delta = (I_C, x)$ berechnen, wobei $x \in \mathbb{R}$ den jeweils notwendigen Wert darstellt. Abb. 42 illustriert diese Herangehensweise.

ÜBER- ODER UNTERSCHREITEN VON SCHRANKENWERTEN

Abs

Der *Abs* Block berechnet den Absolutwert seiner Eingabe I . Die DC fordert, dass mindestens einmal ein negativer und mindestens einmal ein positiver Wert als Eingabe anliegt, es also gilt $I < 0$ bzw. $I \geq 0$.

Dead Zone

Der *Dead Zone* Block besitzt zwei Parameter: *Start of Dead Zone* (b_s) und *End of Dead Zone* (b_e). Beide Schranken müssen zur Erfüllung der DC von der Eingabe I jeweils mindestens einmal über- bzw. unterschritten werden: $I \geq b_s$, $I < b_s$, $I > b_e$ bzw. $I \leq b_e$.

Discrete- Time- Integrator

Der *Discrete-Time-Integrator* besitzt ein externes Reset-Signal I_R , welches zur Erfüllung der DC mindestens einmal aktiviert werden muss. An der Eingabe I_R muss der spezifische Wert 1 anliegen, daher kann wie zuvor beschrieben vorgegangen werden. Das vom Block berechnete Integral (Blockausgabe O) kann darüber hinaus auf einen Minimal- (b_l) und einen Maximalwert (b_u) beschnitten werden (Saturation Limits). Die DC fordert das mindestens einmalige Unter- bzw. Überschreiten beider Schranken: $O \leq b_l$, $O > b_l$, $O \geq b_u$ bzw. $O < b_u$.

For Iterator

Der *For Iterator* führt das Subsystem, in dem es sich befindet, iterativ für die als Blockeingabe I gegebene Anzahl von Durchläufen aus. Als Ausgabe O liefert der Block die Anzahl der bereits

durchlaufenen Iterationen. Die DC fordert, dass das Iterationslimit jeweils mindestens einmal unter- bzw. überschritten wird: $O \leq I$ bzw. $O > I$.

Der Block *Rate Limiter* beschränkt die Änderungen des Eingangssignals I auf einen Minimal- und einen Maximalwert: *Falling Slew Rate* (Δ_{FSR}) bzw. *Rising Slew Rate* (Δ_{RSR}). Die DC fordert, die jeweils mindestens einmalige Unter- bzw. Überschreitung beider Schranken. Die notwendige Steigung Δ_I des Eingangssignals lässt sich aus einzelnen Zeitangaben, dem aktuellen Eingabewert und der Blockausgabe des vorherigen Zeitschritts berechnen. Zur Erfüllung der DC muss gelten: $\Delta_I \geq \Delta_{RSR}$, $\Delta_I < \Delta_{RSR}$, $\Delta_I \leq \Delta_{FSR}$ bzw. $\Delta_I > \Delta_{FSR}$.

Rate Limiter

Der *Relay* Block schaltet je nach Eingabe I und in Abhängigkeit der beiden Schranken *Switch on point* (s_{on}) und *Switch off point* (s_{off}) zwischen zwei Ausgabewerten um. Die DC fordert das jeweils mindestens einmalige Unter- bzw. Überschreiten der beiden Schranken: $I \geq s_{on}$, $I < s_{on}$, $I \leq s_{off}$ bzw. $I > s_{off}$.

Relay

Der *Saturation* Block beschränkt die Eingabe I auf einen Minimal- bzw. Maximalwert: l_{min} bzw. l_{max} . Die DC fordert das jeweils mindestens einmalige Unter- bzw. Überschreiten dieser Schranken: $I \geq l_{max}$, $I < l_{max}$, $I \leq l_{min}$ bzw. $I > l_{min}$.

Saturation

Ein *Triggered Subsystem* ist immer dann aktiv, wenn an der Eingabe I_e ein auslösendes Ereignis vorliegt. Je nach Parametrierung handelt es sich um das Wechseln von einem negativen zu einem positiven Wert, umgedreht oder beides. Die DC fordert das mindestens einmalige Ausführen des Subsystems. Je nach initialem Wert von I_e muss gelten: $I_e \geq 0$ bzw. $I_e < 0$.

Triggered Subsystem

In allen genannten Beispielen ist das Ziel, einen direkt beobachtbaren Datenwert (eine Ein- oder Ausgabe des jeweiligen Blocks) über oder unter einen gegebenen Schrankenwert zu bringen. Die Distanz zur Erfüllung des gewünschten Testziels kann gemäß der Distanzen für Relationsoperatoren ($\delta_{\geq}$, $\delta_{\leq}$, $\delta_{>}$, $\delta_{<}$) berechnet werden. Abb. 43 illustriert diese Herangehensweise. Befindet sich der jeweilige Messwert ($\log_{I_X}$ bzw. $\log_{O_X}$) im gewünschten Bereich, wird der optimale Zielfunktionswert 0 zurückgegeben, anderenfalls der Distanzwert zum Schwellenwert.

Distanzberechnung

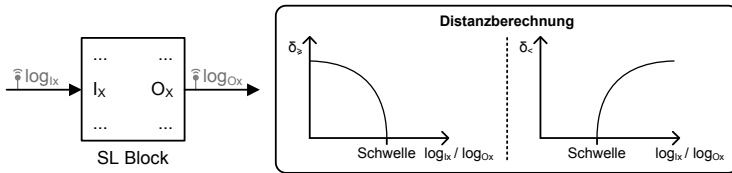


Abb. 43: Distanzberechnung für die Eingabe I_X oder Ausgabe O_X eines Simulink Blocks zur Unter- bzw. Überschreitung eines spezifischen Schwellenwertes. In diesem Fall können die Distanzen wie folgt berechnet werden: $\delta_{\geq} (\log I_X / \log O_X, \text{Schwelle})$ bzw. $\delta_{<} (\log I_X / \log O_X, \text{Schwelle})$.

BEDINGUNG ÜBER MEHRERE EINGABEN

Logical Operator

Der *Logical Operator* verwendet einen logischen Operator $\odot \in \{\text{AND, OR, NAND, NOR, XOR, NOT}\}$ zur Verknüpfung seiner $n \in \mathbb{N}$ Eingaben I_n . Gemäß DC muss der Block jeweils mindestens einmal zu *wahr* und zu *falsch* ausgewertet werden. Es muss gelten: $I_1 \odot I_2 \odot \dots \odot I_n = b$, mit $b \in [1, 0]$. Je nach verwendetem logischen Operator $\odot$ ergeben sich unterschiedliche Bedingungen für alle Eingaben. Gilt beispielsweise $\odot = \text{AND}$ und $b = 0$, ergibt sich die Bedingung $I_1 = 0 \text{ OR } I_2 = 0 \text{ OR } \dots \text{ OR } I_n = 0$. Für die CC muss unabhängig vom verwendeten logischen Operator jede der Eingaben jeweils einmal zu 1 und zu 0 ausgewertet werden. Da es sich bei den Eingaben des Logical Operators um boolesche Signale handelt, müssen die jeweils vorgelagerten Blöcke verwendet werden (vgl. Abb. 40 auf Seite 130). Es ergibt sich für jede Eingabe eine gesonderte Bedingung.

Combinatorial Logic

Der Block *Combinatorial Logic* setzt basierend auf einer frei definierbaren Wahrheitstabelle mehrere Eingaben in Verbindung. Die DC fordert, dass jede Zeile (Kombinationsmöglichkeit der Eingaben) mindestens einmal zur Auswertung kommt. Es ergibt sich eine spezifische Belegung aller Eingaben, die es zu erfüllen gilt. Die CC hingegen fordert die jeweils mindestens einmalige Auswertung aller Eingaben zu *wahr* und *falsch*. Da für diese booleschen Eingaben wieder die vorgelagerten Blöcke verwendet werden müssen, ergibt sich eine zu erfüllende Bedingung.

Enabled Subsystem

Ein *Enabled Subsystem* ist immer dann aktiv, wenn an der Aktivierungseingabe I_a der Wert 1 anliegt. Handelt es sich bei dieser Eingabe um einen Vektor, müssen alle enthaltenen Datenwerte den Wert 1 aufweisen. Die DC fordert das mindestens einmalige

Aktivieren des Subsystems. Die CC hingegen kann nur gemessen werden, wenn es sich bei der Eingabe um einen Vektor handelt: Gefordert ist das jeweils mindestens einmalige Auftreten der Werte 1 und 0 für jeden enthaltenen Datenwert. Da es sich wie beim Logical Operator um einen booleschen Eingabewert handelt, müssen ebenfalls die vorgelagerten SL Blöcke berücksichtigt werden.

Ein *MinMax* Block gibt jeweils den minimalen bzw. maximalen Wert all seiner Eingaben $I_i, i \in \mathbb{N}$ als Ausgabe weiter. Die DC fordert, dass jeweils mindestens einmalige Weiterschalten jeder Eingabe als Minimum bzw. Maximum. Es muss daher für jedes $I_n \in I$ mindestens einmal gelten: $I_n = \min(I)$ bzw. $I_n = \max(I)$.

MinMax

Der *Switch* Operator schaltet in Abhängigkeit seiner booleschen Kontrolleingabe I_k entweder die erste oder die letzte Eingabe weiter. Das mindestens einmalige Weiterschalten beider Eingaben wird von der DC gefordert, d.h. I_k muss einmal zu 1 und einmal zu 0 ausgewertet werden. Da es sich bei I_k um eine boolesche Eingabe handelt, müssen ebenfalls die vorgelagerten SL Blöcke entsprechend berücksichtigt werden, woraus sich eine zu erfüllende Bedingung ergibt.

Switch

Der *While Iterator* ähnelt dem zuvor behandelten For Iterator. Er führt das Subsystem, in dem er sich befindet aus, solange die *While Condition* erfüllt ist. Diese ist als eine boolesche Eingabe gegeben und muss zur Erfüllung der DC mindestens einmal die Werte 1 bzw. 0 aufweisen. Auch in diesem Fall müssen die vorgelagerten SL Blöcke zur Ableitung der notwendigen Bedingung berücksichtigt werden.

*While
Iterator*

Alle vorgenannten Beispiele haben gemein, dass sich eine spezifische, unterschiedlich komplexe Bedingung ergibt, deren Erfüllung zur Erfüllung des Überdeckungsziels führt. Die Distanz zur Erfüllung dieses Überdeckungsziels bzw. dieser Bedingung lässt sich berechnen durch kombinierte Anwendung der Distanzen für Vergleichs- ($\delta_=\, \delta_{\neq}$) und Relationsoperatoren ($\delta_{\geq}, \delta_{\leq}, \delta_{>}, \delta_{<}$) und der Distanzen für logische Verknüpfungen ($\delta_{\&}, \delta_{||}, \delta_{-}$). Abb. 44 illustriert diese Herangehensweise exemplarisch für den SL Block *MinMax*. Dargestellt ist die Distanzberechnung für die Anforderung der DC, Eingabe I_1 als Maximum weiterzuschalten, mitsamt des sich aus der Kombination mehrerer Distanzmaße ergebenden Zielfunktionswerts.

*Distanz-
berechnung*

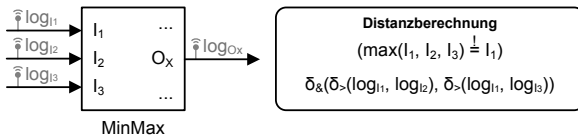


Abb. 44: Berechnung der Distanz zur Erfüllung einer blockspezifischen Bedingung über mehrere Eingaben eines Simulink Blocks beispielsweise für einen MinMax Block und die Bedingung, die erste Eingabe I_1 als Maximum weiterzuschalten. Die Distanz wird durch Verknüpfung mehrerer Distanzoperatoren berechnet.

4.3.2 STATEFLOW DIAGRAMME

Die Überdeckungskriterien CC und DC stellen Anforderungen an die Entscheidungen bzw. deren atomare Bedingungen von Transitionen innerhalb des SF Diagramms. Es ist daher von besonderer Wichtigkeit, einen Distanzwert zum Auslösen einer bestimmten Transition bzw. dem Erfüllen einer der enthaltenen atomaren Bedingungen bestimmen zu können.

Die Distanz zur Auslösung einer Transition wird *Transitionsdistanz* δ_T genannt. Da das Auslösen einer Transition vom Eintreten aller geforderten Transitionsergebnisse und der Gültigkeit aller Transitionsbedingungen abhängt, muss zur Berechnung der Transitionsdistanz sowohl die Distanz zum Eintreten der notwendigen Ereignisse (*Transitionsergebnisdistanz* δ_{TE}) als auch die Distanz zur Gültigkeit aller Transitionsbedingungen (*Transitionsbedingungs-distanz* δ_{TB}) berücksichtigt werden. Zur Bewertung der Annäherung an eine Transitionsauslösung spielt zunächst die Annäherung an den Zustand, von dem die betrachtete Transition ausgeht, eine große Rolle. Diese wird mit Hilfe der *Zustandsannäherungsstufe* δ_{ZAS} bemessen. Aufbauend auf der Zustandsannäherungsstufe und der Transitionsdistanz kann ähnlich zum suchbasierten Strukturtestansatz nach Wegener *et al.* [146] eine Distanz zum Erreichen eines Zustands konstruiert werden – die *Zustandsdistanz* δ_Z .

Abb. 45 illustriert die Abhängigkeit der einzelnen Distanzmetriken, welche in den folgenden Abschnitten erläutert werden.

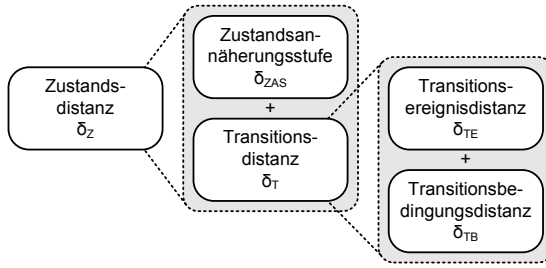


Abb. 45: Abhängigkeit und Zusammenhang der unterschiedlichen für Stateflow relevanten Distanzmetriken.

ZUSTANDSDISTANZ

Die *Zustandsdistanz* beschreibt, wie sehr sich die Ausführung zu einem bestimmten Zeitschritt dem Zielzustand z_z angenähert hat. Sie berechnet sich analog zum Ansatz von Wegener *et al.* [146], da sie ebenfalls sowohl ein Annäherungslevel zum Zielzustand als auch ein Analogon zur Zweigdistanz – die Transitionsdistanz – beinhaltet.

Während im Ansatz von Wegener *et al.* die Ausführungspfade innerhalb des Kontrollflussgraphen des Testobjekts untersucht werden, kommt in dieser Arbeit ein Satz von Pfaden zum Zielzustand innerhalb des SF Zustandsdiagramms zur Berechnung der Annäherung zum Einsatz. Ein Pfad von einem aktiven Zustand z_a zu Zielzustand z_z ist gegeben als eine Aneinanderreihung aller auf diesem Pfad liegenden Zustände:

$$p_n^{z_a z_z} = \langle z_a, \dots, z_z \rangle. \quad (4.2)$$

Zu jedem Zeitschritt können aufgrund der Parallelität und Hierarchie des Zustandsdiagramms mehrere Zustände gleichzeitig aktiv sein. Diese müssen jeweils in die Distanzberechnung einbezogen werden: Es müssen alle Pfade von allen aktiven Zuständen zum Zielzustand bestimmt und separat berücksichtigt werden. Der Bezeichner $P^{z_a z_z}$ beschreibt die Menge aller Pfade ausgehend von allen $n \in \mathbb{N}$ aktiven Zuständen zum Zielzustand z_z :

$$P^{z_a z_z} = \{p_n^{z_a z_z} | n \in \mathbb{N}\}. \quad (4.3)$$

Schließlich bezeichnet P_z^z die Menge aller Pfade, die von einem der aktiven Zustände ausgehend, zum Zielzustand verlaufen.

Es handelt es sich um die Vereinigung der Mengen $P^{z_a z_z}$ aller aktiven Zustände $z_a \in Z_a$:

$$P^{z_z} = \bigcup_{z_a \in Z_a} P^{z_a z_z}. \quad (4.4)$$

Definition 4.1. Die *Zustandsdistanz* $\delta_Z : Z \times \mathbb{R} \rightarrow \mathbb{R}_{[0,1]}$ bemisst die Annäherung an einen Zielzustand $z_z \in S$ im Zustandsdiagramm. Sie ist definiert als die minimale Distanz aller in Zeitschritt $t \in \mathbb{R}$ aktiven Zustände $z_a \in Z_a \subset Z$ zu z_z und berechnet sich als die Summe der Zustandsannäherungsstufe δ_{ZAS} und der Transitionsdistanz δ_T :

$$\begin{aligned} \delta_Z(z_z)(t) &= \min_{z_a \in Z_a} \left(\widetilde{\delta}_Z(P^{z_a z_z})(t) \right), \\ \widetilde{\delta}_Z(p)(t) &= \text{norm}(\delta_{ZAS}(p)(t) + \delta_T(\tau)(t)). \end{aligned} \quad (4.5)$$

Die beiden Distanzmetriken δ_{ZAS} und δ_T realisieren eine Ziel-funktionskonstruktion analog zum Ansatz von Wegener *et al.* [146]: Die Zustandsannäherungsstufe liefert eine grobe Annäherung an den Zielzustand und die Transitionsdistanz bemisst die Distanz zum Auslösen der ersten abweichenden Transition τ auf dem kürzesten Pfad zum Zielzustand. Auf beide Distanzmetriken wird in den folgenden Abschnitten genauer eingegangen. Die Funktion $\text{norm} : \mathbb{R} \rightarrow \mathbb{R}_{[0,1]}$ normiert die Eingabe auf das Intervall $[0, 1]$ (vgl. Abschnitt 2.2.2 auf Seite 31).

ZUSTANDSANNÄHERUNGSSTUFE

Die Zustandsannäherungsstufe ist ein grobes Distanzmaß für die Annäherung an einen gewünschten Zielzustand innerhalb des Zustandsdiagramms. Es wird bemessen an der Anzahl der Zustände, die zwischen einem aktiven Zustand und dem Zielzustand liegen.

Definition 4.2. Die *Zustandsannäherungsstufe* $\delta_{ZAS} : P \rightarrow \mathbb{N}$ bemisst die Annäherung eines Zustands zu einem anderen Zustand. Bemessungsgrundlage ist ein Pfad p vom Initial- zum Endzustand:

$$\delta_{ZAS}(p) = |p| - 2 \quad (4.6)$$

Für einen gegebenen Pfad p zwischen zwei Zuständen, wird die Zustandsannäherungsstufe als deren Distanz in Form der Anzahl der Zustände oder Verzweigungspunkte zwischen ihnen spezifiziert. Nicht enthalten sind der Initial- und der Endzustand.

TRANSITIONSDISTANZ

Die Transitionsdistanz ist ein für eine Transition t spezifisches Distanzmaß zur Bemessung der Annäherung an die Ausführung dieser Transition. Vorausgesetzt der Ursprungszustand der Transition ist aktiv, ist die Transition genau dann gültig und wird ausgeführt, wenn ihre auslösenden Ereignisse auftreten und ihre Transitionsbedingung erfüllt ist. In die Berechnung der Transitionsdistanz muss daher sowohl die Distanz zu den auslösenden Ereignissen als auch die Distanz zu ihrer Transitionsbedingung einfließen.

Definition 4.3. Die *Transitionsdistanz* $\delta_T : T \times \mathbb{R} \rightarrow \mathbb{R}_{[0,1]}$ bemisst die Annäherung an das Auslösen einer Transition $\tau \in T$ zum Zeitpunkt $t \in \mathbb{R}$. Sie ist definiert als die Konjunktion der beiden Distanzen Transition ereignisdistanz δ_{TE} und Transitionsbedingungsdi-
stanz δ_{TB} :

$$\delta_T(\tau)(t) = \text{norm}(\delta_{\&}(\delta_{TE}(\tau)(t), \delta_{TB}(\tau)(t))) \quad (4.7)$$

Die Transition ereignisdistanz d_{TE} bemisst die Annäherung an die Transition ereignisse, während die Transitionsbedingungsdi-
stanz d_{TC} die Annäherung an die Erfüllung der Transitionsbe-
dingung bemisst – diese beiden Distanzmetriken werden in den
folgenden Abschnitten vorgestellt.

TRANSITIONSEREIGNISDISTANZ

Die Transition ereignisdistanz ist ein Maß für die Annäherung an das Auftreten eines Ereignisses e bzw. mehrere Ereignisse, die zur Auslösung der betrachteten Transition notwendig sind. Da Transitionen von einer beliebigen Anzahl von Ereignissen abhängen können, muss die Distanz zu jedem einzelnen Ereignis separat berechnet und anschließend gemäß der Distanzfunktion der Konjunktion ($\delta_{\&}$) kombiniert werden.

Definition 4.4. Die *Transition ereignisdistanz* $\delta_{TE} : T \times \mathbb{R} \rightarrow \mathbb{R}_{[0,1]}$ bemisst die Annäherung an das Eintreten aller Ereignisse $e \in E_\tau$, welche zur Auslösung der Transition τ notwendig sind:

$$\delta_{TE}(\tau)(t) = \text{norm} \left(\delta_{\&} \left(\widetilde{\delta_{TE}}(e)(t) \right) \right). \quad (4.8)$$

SF Ereignisse können an unterschiedlichen Stellen (Auslösepunkten) des Zustandsdiagramms ausgelöst werden. Die Distanz $\widetilde{\delta}_{TE}$ zum Auslösen der einzelnen atomaren Ereignisse e muss in Abhängigkeit vom jeweiligen Auslösepunkt bestimmt werden. Im Folgenden sind die unterschiedlichen Typen der Auslösepunkte und die notwendigen Herangehensweisen beschrieben.

CONDITION ACTION Das Zielereignis e wird innerhalb einer Condition Action einer Transition ausgelöst. In diesem Fall setzt sich die Distanz zusammen aus dem Erreichen des Ursprungszustands z_{τ_e} der Zielereignis-auslösenden Transition τ_e und der Erfüllung seiner Transitionsbedingung ohne jedoch notwendigerweise seine Ereignisbedingung zu erfüllen.

$$\widetilde{\delta}_{TE}(e)(t) = \begin{cases} 1 + \delta_{TB}(\tau_e)(t) & \text{falls } \delta_Z(z_{\tau_e})(t) = 0, \\ \delta_Z(z_{\tau_e})(t) & \text{sonst.} \end{cases} \quad (4.9)$$

Als Transitionsergebnisdistanz wird die um den Wert 1 inkrementierte Zustandsdistanz δ_Z zu Zustand z_{τ_e} zurückgegeben. Ist dieser Zustand bereits aktiv, muss lediglich die Transitionsbedingungs-distanz δ_{TB} der Transition τ_e zurückgegeben werden. Die Zustandsdistanz wird inkrementiert, um sicherzustellen, dass der kleinstmögliche Zielfunktionswert bei Nichterreichen von Zustand z_{τ_e} in jedem Fall größer ist, als die Transitionsbedingungs-distanz bei erreichtem Zustand z_{τ_e} .

TRANSITION ACTION Wird das Zielereignis e innerhalb einer Transition Action einer Transition ausgelöst, muss die Distanz zum Auslösen dieser Transition bestimmt werden. Transition Actions werden genau dann ausgelöst, wenn sowohl das Transitionsergebnis eintritt und die Transitionsbedingung erfüllt ist. Deshalb kann die Distanz berechnet werden als die Distanz zum Erreichen des Zielzustandes z_{τ_e} dieser Transition.

$$\widetilde{\delta}_{TE}(e)(t) = \delta_Z(z_{\tau_e})(t). \quad (4.10)$$

Dies schließt den Fall aus, dass der Zielzustand z_{τ_e} bereits aktiv ist – die Zustandsdistanz würde in diesem Fall den optimalen Zielfunktionswert liefern. Innerhalb der Zielfunktionsberechnung dürfen deshalb nicht die einfachen Pfade

zum Zielzustand gegeben sein, sondern jene Pfade, die aus diesem Zustand heraus- und wieder in diesen Zustand einführen.

STATE ACTION Wird das Zielereignis e innerhalb einer State Action eines Zustands z_e ausgelöst, kann die Distanz zur Auslösung dieses Ereignisses ähnlich wie die Zustandsdistanz zu z_e berechnet werden:

$$\widetilde{\delta_{TE}}(e)(t) = \delta_Z^*(P)(t). \quad (4.11)$$

Die Menge Z_e umfasst alle Zustände, in deren State Action das Ereignis e ausgelöst wird. Die Funktion δ_Z^* ähnelt der Zustandsdistanz δ_Z . Der einzige Unterschied besteht darin, dass kein Zielzustand angegeben wird, sondern statt dessen ein einzelner Satz von Pfaden P , für den die Distanzberechnung ansonsten identisch verläuft.

Es existieren in Abhängigkeit des Typs der State Action unterschiedliche Herangehensweisen:

ENTRY Das Zielereignis wird innerhalb einer *entry* state action ausgelöst, d.h. es wird immer dann ausgelöst, wenn der Zustand betreten wird. Die Menge P enthält alle Pfade ausgehend von allen aktiven Zuständen $z_a \in Z_a$ zu den Zuständen Z_e . Falls jedoch einer dieser Zustände bereits aktiv ist, muss für diesen der jeweils kürzeste Pfad aus dem Zuständen heraus und wieder hinein angegeben werden.

DURING Wird das Zielereignis innerhalb einer *during* state action ausgelöst, kann die Distanz zum Auslösen dieser State Action durch die Distanz zu diesem Zustand ausgedrückt werden, da es immer dann ausgelöst wird, wenn der Zustand aktiv ist. P enthält in diesem Fall alle von allen aktiven Zuständen $z_a \in Z_a$ ausgehenden Pfade, die zu den Zuständen Z_e verlaufen.

EXIT Im Unterschied zu *entry* state actions werden *exit* state actions immer dann ausgelöst, wenn der Zustand verlassen wird. Aus diesem Grund muss die Menge P alle Pfade enthalten, die die Zustände Z_e verlassen bzw. falls diese Zustände inaktiv sind, der jeweils kürzeste Pfad von allen aktiven Zuständen $z_a \in Z_a$ ausgehend in diese Zustände hinein und wieder heraus.

FROM SIMULINK Das Zielereignis e wird außerhalb des SF Diagramms ausgelöst. In diesem Fall muss dessen Ursprung bestimmt werden. Besitzt das Ereignissignal keine Verbindung zu einer optimierten Modelleingabe, z.B. falls es einem statischen Impulsgenerator entspringt, kann die Distanz nicht berechnet werden und der bestmögliche Zielfunktionswert muss zurückgegeben werden: $\widetilde{\delta_{TE}}(e)(t) = 0$. Falls nicht, muss die Distanz zum Erreichen des Wertes 1 in Abhängigkeit des verwendeten SL Blocks SB berechnet werden: $\delta_{TE}(e)(t) = d_=(SB, 1.0)(t)$.

TRANSITIONSBEDINGUNGSDISTANZ

Die Transitionsbedingungsdistanz ist ein Maß für die Annäherung an das Erfüllen einer Bedingung c bzw. mehrerer Bedingungen, was zur Auslösung der betrachteten Transition notwendig ist. Da Transitionen von einer beliebigen Anzahl von Transitionsbedingungen abhängen können, muss die Distanz zu jeder einzelnen Bedingung separat berechnet und anschließend gemäß der Distanzfunktion der Konjunktion ($\delta_{\&}$) kombiniert werden.

Definition 4.5. Die *Transitionsbedingungsdistanz* $\delta_{TB} : T \times \mathbb{R} \rightarrow \mathbb{R}_{[0,1]}$ bemisst die Annäherung an das Erfüllen aller Bedingungen $c \in C_\tau$, welche zur Auslösung der Transition τ notwendig sind:

$$\delta_{TB}(\tau)(t) = \text{norm} \left(\delta_{\&} \left(\widetilde{\delta_{TB}}(c)(t) \right) \right)_{c \in C_\tau}. \quad (4.12)$$

Transitionsbedingungen können unter Verwendung unterschiedlicher Techniken definiert werden. Die Distanz $\widetilde{\delta_{TB}}$ zum Erfüllen der einzelnen Bedingungen c muss in Abhängigkeit von der jeweils verwendeten Technik bestimmt werden. Im Folgenden sind diese unterschiedlichen Techniken und die notwendigen Herangehensweisen beschrieben.

BOOLEAN OPERATORS Falls die zu erfüllende Bedingung aus booleschen Ausdrücken besteht, die Variablen untereinander und mit einzelnen Datenwerten in Relation setzen, können die Distanzmetriken für Vergleichs- und Relationsoperatoren und jene für logische Verknüpfungen direkt verwendet werden (vgl. Abschnitt 2.2.2 auf Seite 31).

IN(STATENAME) SF stellt eine spezielle Funktion zur Verfügung, welche zu *wahr* ausgewertet wird, wenn der als Argument übergebene Zustand z_c aktiv ist. Um diese Bedingung zu erfüllen, muss die Distanz zur Erfüllung dieses speziellen Zustands als Transitionsbedingungsdistanz zurückgegeben werden.

$$\widetilde{\delta}_{TB}(c)(t) = \delta_Z(z_c)(t) \quad (4.13)$$

TEMPORAL LOGIC SF Bedingungen können auch auf temporalen Logikoperatoren basieren, welche zeitliche Anforderungen an das Auftreten von SF Ereignissen stellen: $\Theta(n, e)$, wobei $\Theta \in \{after, at, before, every\}$. Jedem der Operatoren ist ein Referenzwert n und ein Bezugsereignis e zugeordnet.

Zur Bestimmung eines Zielfunktionswertes, muss die Anzahl des Auftretens $\#e$ von Ereignis e vom Zeitschritt der letzten Aktivierung des assoziierten Zustands bis zum derzeitigen Zeitschritt t gezählt werden. In Abhängigkeit des jeweiligen temporalen Operators und des Referenzwertes n , werden unterschiedliche Distanzmetriken angewandt:

$$\widetilde{\delta}_{TB}(c)(t) = \begin{cases} \delta_{\geq}(\#e, n) & \Theta = after, \\ \delta_{= }(\#e, n) & \Theta = at, \\ \delta_{<}(\#e, n) & \Theta = before, \\ \delta_{= }(\#e \bmod n, n) & \Theta = every. \end{cases} \quad (4.14)$$

Der *after* Operator wird zu *wahr* ausgewertet, sobald das Bezugsereignis e mindestens n Mal auftritt, d.h. $\#e \geq n$. Im Gegensatz dazu wird der *before* Operator solange zu *wahr* ausgewertet, bis die Anzahl des Auftretens des Bezugsereignisses kleiner ist als der gegebene Referenzwert, d.h. $\#e < n$. Der *at* Operator ist genau dann erfüllt, wenn das Bezugsereignis e zum n 'ten Mal auftritt. Die Distanz zur Erfüllung der Bedingung $\#e == n$ wird als Distanz für diesen Operator verwendet. Der *every* Operator ist *wahr* bei jedem n 'ten Auftreten des Bezugsereignisses seit Aktivierung des assoziierten Zustands, d.h. $\#e \bmod n == n$.

CALL TO FUNCTIONS SF Bedingungen können ebenfalls auf den Ausgaben von Funktionen basieren. Beispiele dafür sind, in

Form von Flussdiagrammen gegebene (grafische) Funktionen, ML Funktionen oder eingebettete C-Funktionen. Um die Suche ausreichend steuern zu können, müssen diese Funktionen zunächst statisch analysiert werden. Dies ermöglicht erst die Bestimmung, welcher Zweig bzw. Pfad durchlaufen werden muss, um eine bestimmte Ausgabe hervorzurufen. Ist diese Information gegeben, kann die Distanz zu diesen Zweigen oder Pfaden mit Hilfe gängiger suchbasierter Strukturtestverfahren bestimmt werden. Dies überschreitet jedoch mitsamt der notwendigen statischen Analysen den Rahmen dieser Arbeit und wird aus diesem Grund hier nicht weiter betrachtet.

4.4 ZUSAMMENFASSUNG

Dieses Kapitel befasste sich mit der Überdeckung von ML/SL/SF Modellen. Es wurde eine Möglichkeit vorgestellt, wie für diese Modelle generierte Teststimuli hinsichtlich ihrer Eignung, ein gesuchtes strukturelles Überdeckungsziel zu erreichen, bewertet werden können.

Über-
deckungs-
kriterien

Zunächst wurde in Abschnitt 4.1 beschrieben, wie die gängigsten Überdeckungskriterien auf SL/SF angewandt werden können. Alle für diese Kriterien relevanten strukturellen Elemente wurden vorgestellt und alle zur Bewertung der Teststimuli notwendigen Beobachtungen während der Ausführung des Testobjekts identifiziert.

Instrumen-
tierung

Basierend darauf wurde in Abschnitt 4.2 vorgestellt, wie SL Blöcke bzw. SF Diagramme instrumentiert werden können, um eben diese Beobachtungen während der Testobjektausführung zu ermöglichen. Dafür kann für SL Blöcke die *Simulink Signal Logging* Funktionalität und für SF Diagramme die *Stateflow Signal Logging* Funktionalität verwendet werden. Einzig zur Protokollierung der Transitionen eines SF Diagramms muss eine gesonderte Instrumentierung vorgenommen werden.

Distanz-
metriken

Die protokollierten Beobachtungen werden anschließend zur Berechnung einer Bewertung für die Teststimuli verwendet. Zu diesem Zweck kommen dedizierte Distanzmetriken für SL und SF zum Einsatz, welche die Distanz zur Erfüllung des gesuchten

strukturellen Überdeckungsziels quantifizieren. In Abschnitt 4.3 wurden die für SL und SF spezifischen Distanzmetriken vorgestellt. Die für SL Blöcke notwendige Distanzberechnung kann in drei Klassen unterteilt werden. Für jede der drei Klassen wurden Beispiele genannt und aufgezeigt, wie eine Distanzberechnung durchgeführt werden kann. Zur Bestimmung von Distanzmaßen für SF Diagramme wurde ein Ansatz ähnlich dem Annäherungsansatz von Wegener *et al.* [146] vorgestellt. Dieser verwendet diverse dedizierte Distanzmaße, welche im Detail beschrieben wurden.

5 | FALLSTUDIE

*Nur ein Narr macht
keine Experimente.*

— Charles Darwin

Dieses Kapitel beschreibt die empirische Fallstudie zur Bewertung des in dieser Arbeit entwickelten Testdatengenerierungsverfahrens. Die Arbeit zielt auf die Behebung der Einschränkungen vorhandener Testdatengenerierungsverfahren ab, welche bei der Verwendung für komplexe ML/SL/SF Modelle auftreten. Ob dies gelungen ist, soll durch diese Fallstudie experimentell überprüft werden.

Eine empirische Fallstudie ist im Wesentlichen ein Vergleich von Annahmen mit Beobachtungen. Die Essenz einer empirischen Fallstudie ist das Bestreben, Schlüsse aus dem Vergleich von Theorie und Praxis zu ziehen, um in Folge dessen den Untersuchungsgegenstand verbessern zu können. Laut Kitchenham *et al.* [77] und Perry *et al.* [106] sollte eine empirische Fallstudie die folgenden Komponenten enthalten:

*Konzeption
empirischer
Fallstudien*

- A. Den experimentellen Kontext der Fallstudie,
- B. den Aufbau der Fallstudie,
- C. das Durchführen von Experimenten mitsamt einer sinnvollen Datenerhebung,
- D. die Analyse der erhobenen Daten und
- E. die Interpretation und Präsentation der Ergebnisse, Bezugnehmend auf die formulierte Forschungsfrage.

Der experimentelle Kontext der Fallstudie ist gegeben durch den aktuellen Stand der Technik im Bereich des suchbasierten Strukturtests und wurde in Kapitel 2 auf Seite 11 ausführlich erläutert. Dieses Kapitel beschreibt die restlichen Komponenten: Abschnitt 5.1 beschreibt zunächst die technische Realisation des entwickelten Verfahrens. Abschnitt 5.2 geht auf den Aufbau der Fallstudie

Gliederung

(Komponente B) ein. Abschnitt 5.3 hingegen befasst sich mit der Durchführung der Experimente, dem Erfassen von Daten und deren Analyse und Interpretation (Komponenten C, D und E).

5.1 REALISIERUNG DES VERFAHRENS

Dieser Abschnitt beschreibt die technische Realisation des in dieser Arbeit entwickelten Verfahrens zur suchbasierten Testdatengenerierung für SL Modelle. Dieses Verfahren ist im Testdatengenerator *SBGen-SL* implementiert.

IMPLEMENTIERUNG UND ABLAUF

<i>Allgemeine Funktion</i>	Abb. 46 beschreibt die Systemarchitektur von <i>SBGen-SL</i> . Eingabe in das System ist die vom Benutzer bereitgestellte Testspezifikation, bestehend aus der Spezifikation des Testobjekts und weiteren testspezifischen Angaben, einer Signalspezifikation (<i>Signal Specification</i>) sowie ggf. einer Constraint Spezifikation (<i>Constraint Specification</i>). Weitere testspezifische Angaben umfassen beispielsweise das zu erfüllende strukturelle Überdeckungskriterium oder die Konfiguration des Suchverfahrens.
<i>Hauptkomponenten</i>	Das System kann grob in drei Hauptkomponenten unterteilt werden: <i>Teststeuerung</i> , <i>Optimierung</i> und <i>Testobjektanalyse und -interaktion</i> . Die Teststeuerung (Komponente <i>Test Control</i>) steuert den gesamten Arbeitsablauf. Basierend auf der vom Benutzer zur Verfügung gestellten Testspezifikation veranlasst sie die Testvorbereitung und verwendet die Komponente Optimierung zur Durchführung der Suche nach relevanten Testdaten. Die Komponenten und der zugrundeliegende Workflow wird im Folgenden genauer beschrieben.
<i>Testvorbereitung</i>	Der gesamte Testdatengenerierungsprozess wird nach Eingabe der Testspezifikation vom Benutzer gestartet. Die Teststeuerung stößt daraufhin die Analyse und Instrumentierung des Testobjekts (<i>Analyzer & Instrumenter</i>) an. Dies geschieht über den Aufruf von direkt in ML integrierten Befehlsabfolgen (ML Scripts). Diese untersuchen das SL Modell in Abhängigkeit des vom Benutzer gewünschten strukturellen Überdeckungskriteriums und bestimmen somit die zu erreichenden Überdeckungsziele. Diese

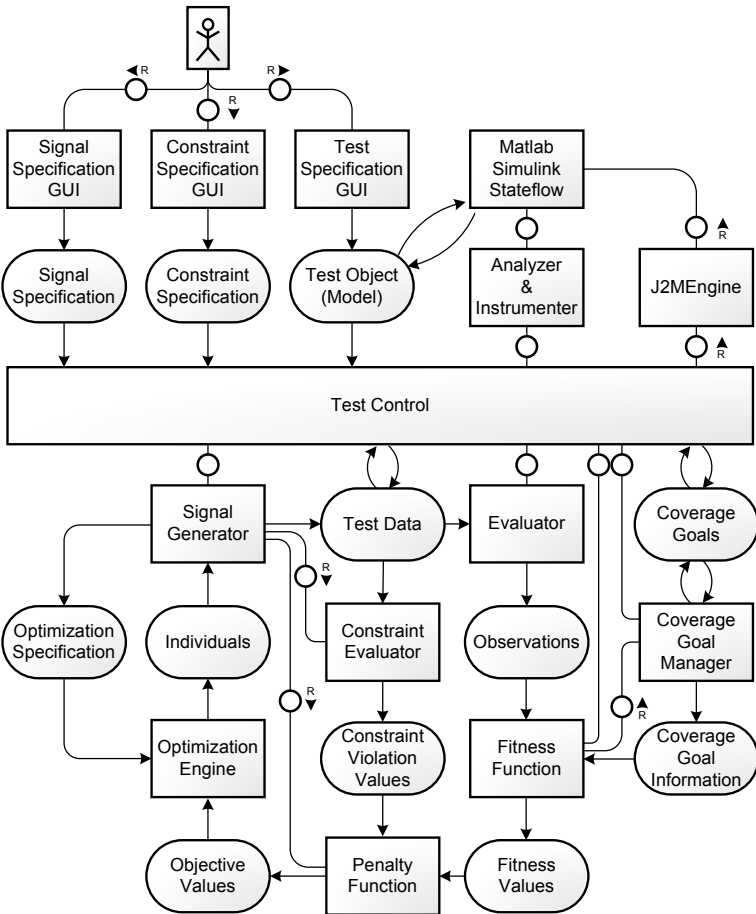


Abb. 46: SBGen-SL: Systemarchitektur zur Realisation des entwickelten Verfahrens

berücksichtigend, wird das SL Modell mitsamt aller enthaltenen SF Diagramme instrumentiert. Nachdem das Testobjekt somit für die beobachtbare Ausführung vorbereitet wurde, initiiert die Teststeuerung den suchbasierten Testdatengenerierungsprozess. Der *Coverage Goal Manager* verwaltet die zu erreichenden Überdeckungsziele. Für jedes noch nicht erreichte Überdeckungsziel wird iterativ eine separate Optimierung durchgeführt. Wird auf der Suche nach einem Überdeckungsziel ein anderes Überdeckungsziel erreicht, muss für dieses anschließend keine separate Suche mehr

durchgeführt werden. Für jedes Überdeckungsziel, für das eine Suche durchgeführt werden soll, wird der Suchprozess wie folgt eingeleitet.

Initialisierung der Suche Der *Signal Generator* und der *Constraint Evaluator* werden zunächst mit den Informationen aus der Testspezifikation initialisiert. Der *Signal Generator* erstellt seinerseits die für das verwendete Optimierungsverfahren (*Optimization Engine*) notwendige *Problem Specification*. Das Optimierungsverfahren ist nun für die Erstellung genotypischer Individuen bereit. Der *Constraint Evaluator* wird derweil hinsichtlich der spezifizierten Constraints initialisiert.

Generierung der Testdaten Der Suchprozess beginnt mit der Generierung der ersten Population genotypischer Individuen, welche dem *Signal Generator* vom Optimierungsverfahren übergeben werden. Der *Signalgenerator* dekodiert diese in eine Population phänotypischer Individuen (Testdaten). Die Testdaten werden zunächst vom *Constraint Evaluator* hinsichtlich der Erfüllung der spezifizierten Constraints untersucht. Die berechneten Verletzungsmaße (*Constraint Violation Values*) werden zwischengespeichert.

Ausführung der Testdaten Alle keine kritischen Constraints verletzenden Testdaten werden nun vom *Evaluator* zur Ausführung des Testobjekts verwendet. Dazu werden die Testdaten mit Hilfe der *J2MEngine* iterativ nach ML übertragen. Nach der Ausführung werden ebenfalls auf diesem Weg die Protokolle des Daten- und Kontrollflusses an den *Evaluator* zurückgespielt.

Bewertung der Testdaten Der *Coverage Goal Manager* stellt für jedes Überdeckungsziel Informationen über die Konstruktion der Fitnessfunktion bereit. Die dazu erforderlichen Daten werden den Ausführungsprotokollen entnommen. Für jedes noch zu erreichende Überdeckungsziel wird so die Annäherung an dessen Ausführung ermittelt. Jedes Überdeckungsziel, für das der optimale Fitnesswert erreicht wurde, wird vom *Coverage Goal Manager* als erreicht markiert. Die berechneten Fitnesswerte aller Testdaten bzgl. des aktuellen Überdeckungsziels wird an die *Penalty Funktion* übergeben, welche diese mit den zuvor berechneten Verletzungsmaßen für die spezifizierten Constraints kombiniert. Die so berechneten finalen Fitnesswerte werden als Bewertung der generierten Individuen an das Optimierungsverfahren übergeben. Das Optimierungsverfahren generiert auf diesen Bewertungen basierend eine neue Generation genotypischer Individuen und der Kreislauf wird fortgesetzt. Dies wird so lange wiederholt, bis ein das aktuelle

Überdeckungsziel erreichendes Testdatum gefunden wurde oder aber das konfigurierte Abbruchkriterium erfüllt ist.

5.2 AUFBAU DER FALLSTUDIE

Dieser Abschnitt beschreibt den Aufbau der empirischen Fallstudie. Dazu zählt zum einen die Formulierung der Forschungsfrage und ihre Verfeinerung in überprüfbare Propositionen. Zum anderen zählt dazu das Bestimmen von Kriterien für die Auswahl der zu verwendenden Testobjekte und die Beschreibung des Versuchsaufbaus. Ferner wird die geplante Datenerhebung beschrieben und es werden Kriterien zur Interpretation der experimentellen Daten aufgestellt. Abschließend wird auf die möglichen Bedrohungen der Validität der Fallstudie eingegangen.

Die folgenden Abschnitte gehen im Detail auf diese einzelnen Komponenten der Fallstudie ein.

5.2.1 FORSCHUNGSFRAGE

Wie aus der Beschreibung des Stands der Technik in Kapitel 2 auf Seite 11 ersichtlich, ist keiner der gegebenen Ansätze zur Automatisierung der Testdatengenerierung dazu geeignet, erfolgreich im industriellen Umfeld für SL/SF Modelle angewandt zu werden. Diese existierenden statischen bzw. dynamischen Verfahren stoßen bei komplexen Testobjekten gleichermaßen an Ihre Grenzen (vgl. Abschnitt 2.2.4 auf Seite 44).

Das Ziel dieser Arbeit ist es, diese Grenzen zu überwinden. Die zu untersuchende Forschungsfrage dieser Arbeit lautet demnach:

„Kann die strukturorientierte Testdatengenerierung für komplexe SL/SF Modelle durch Anwendung des suchbasierten Strukturtests erfolgreich automatisiert werden?“

Um diese Frage zu beantworten, muss eine Verfeinerung in einzelnen Propositionen durchgeführt werden, welche mit Hilfe diverser Messwerte der Experimente beurteilt werden können. Die folgenden Propositionen P₁, P₂ und P₃ betrachten die relevantesten Einflussfaktoren:

- P1 Das entwickelte Verfahren weist im Vergleich zu existierenden Testdatengenerierungsansätzen – gemessen an der erreichten strukturellen Überdeckung des Testobjekts – eine höhere *Effektivität* der Testdatengenerierung auf.
- P2 Das entwickelte Verfahren weist im Vergleich zu existierenden Testdatengenerierungsansätzen – gemessen am zeitlichen Aufwand – eine höhere *Effizienz* der Testdatengenerierung auf.
- P3 Bei den erzeugten Eingabedaten handelt es sich um *plausible Testdaten* für das jeweilige Testobjekt.

5.2.2 TESTOBJEKTAUSWAHL

KRITERIEN ZUR AUSWAHL DER TESTOBJEKTE

Die folgenden Kriterien wurden bei der Auswahl der untersuchten Testobjekte berücksichtigt:

VERFÜGBARKEIT Die untersuchten SL Modelle sollen frei verfügbar sein, um die Reproduzierbarkeit der Experimente zu gewährleisten. Dies ermöglicht Vergleiche für zukünftige Arbeiten.

STIMULIERBARKEIT Die Testobjekte sollen mindestens eine Eingabe besitzen, für die ein geeigneter Stimulus generiert wird.

REELLWERTIGE EINGABEN Die Eingaben des Testobjekts sollen möglichst reellwertig sein, um dadurch die Ansprüche an die verwendeten Testdatengenerierungsverfahren zu erhöhen.

REALITÄTSNÄHE Die Testobjekte sollen möglichst realistisch sein, um so einen Einblick in die Einsatzfähigkeit der Testdatengenerierungsverfahren für reale Problemstellungen zu erhalten.

KOMPLEXITÄT Die Testobjekte sollen einen gewissen Komplexitätsgrad aufweisen, d.h. die Überdeckung der enthaltenen strukturellen Elemente sollte nicht trivial sein.

AUSGEWÄHLTE TESTOBJEKTE

Die im Folgenden vorgestellten Testobjekte wurden für die Fallstudie ausgewählt.

- TO1 *Auto Climate* – Dieses Testobjekt modelliert eine automatische Fahrzeugklimaanlage. Das Modell ist ein Beispielm-
odell der SL Dokumentation [132].
- TO2 *Fuel Control* – Dieses Testobjekt modelliert ein fehlertolerantes Kraftstoffeinspritzsystem. Das Modell ist ein Beispielm-
odell der SL Dokumentation [134].
- TO3 *Auto Transmission* – Dieses Testobjekt modelliert die Steuerung eines Automatikgetriebes. Das Modell ist ein Beispielm-
odell der SL Dokumentation [133] und wurde in dieser Arbeit bereits mehrfach referenziert (vgl. Abschnitt 2.1.2).
- TO4 *Daimler Wischer* – Dieses Testobjekt modelliert die Steuerung eines Fahrzeugscheibenwischers. Bei dem Modell handelt es
sich um ein reales Modell von Daimler aus dem Jahr 2002.

Diese Testobjekte wurden den Auswahlkriterien entsprechend ausgewählt. Jedes der Testobjekte erfüllt die Auswahlkriterien der Simulierbarkeit und der Verwendung reellwertiger Eingaben. Die Auswahlkriterien Verfügbarkeit und Realitätsnähe konnten leider nicht gleichzeitig erfüllt werden. Die Testobjekte TO1, TO2 und TO3 sind frei verfügbar, da es sich um offizielle Beispielm-
odelle aus der SL Dokumentation handelt. Aus diesem Grund sind sie jedoch in vielerlei Hinsicht vereinfacht aufgebaut, was zu einer geringeren Realitätsnähe der Modelle führt. Im Gegensatz handelt es sich bei Testobjekt TO4 um ein reales Modell der Daimler AG, welches in vergleichbaren Versionen in Fahrzeugen der Marke Mercedes verbaut wurde. Aus Gründen der Geheimhaltung ist dieses jedoch nicht frei verfügbar; das Auswahlkriterium der Verfügbarkeit somit nicht erfüllt. Bei allen Testobjekten handelt es sich um komplexe Modelle, d.h. die vollständige Überdeckung aller strukturellen Elemente ist nicht trivial.

*Auswahl-
begründung*

Tab. 9 gibt einen Überblick über unterschiedliche Modelleigenschaften der Testobjekte, welche ebenfalls Rückschlüsse auf die Komplexität selbiger erlauben. Gemessen an diesen Modelleigenschaften weist das reale Testobjekt TO4 erwartungsgemäß die höchste Komplexität auf. Die aufgeführten Modelleigenschaften

*Modellei-
genschaften*

Modelleigenschaft		TO ₁	TO ₂	TO ₃	TO ₄
Simulink	Systemeingaben	2	1	2	7
	Blöcke	117	230	69	642
	Subsysteme	10	24	6	39
Stateflow	SF Diagramme	1	1	1	12
	Zustände	19	30	15	40
	Transitionen	26	34	19	246
Überdeckungsziele (Bedingungsüb.)	Insgesamt	58	72	12	381
	Erreichbar	43	38	12	232
	Nicht erreichbar	15	34	0	149

Tab. 9: Modelleigenschaften der ausgewählten Testobjekte.

beinhalten bereits die in den folgenden Abschnitten beschriebenen, zur Ausführung der Testobjekte notwendigen Anpassungen. Dazu zählt zum einen das ggf. notwendige zur Verfügung stellen von Systemeingaben. Zum anderen zählen dazu Modelltransformationen, welche notwendig waren, um das jeweilige Modell für das kommerzielle Werkzeug Reactis Tester anwendbar zu machen. Aufgrund der fehlenden Unterstützung diverser SL Blöcke, mussten diese mit Hilfe von unterstützten Blöcken – ohne Änderung der Semantik – ersetzt werden.

MODELLSPEZIFIKA: TO1

*System-
eingaben*

Das Modell besitzt zwei fest kodierte Parameter */User Setpoint* (Eingabe E1) und */External Temperature* (Eingabe E2), welche die vom Fahrer ausgewählte Innenraumtemperatur bzw. die aktuelle Außentemperatur angibt. Diese Konstanten wurden durch reellwertige Systemeingaben ersetzt. In beiden Fällen werden Temperaturen im Bereich zwischen -99°C und 100°C erwartet.

*Anpas-
sungen für
Reactis*

Das ursprüngliche Modell verwendet einen zeitbasierten Pulsgenerator (*/System Trigger*). Da dieser nicht unterstützt wird, musste er durch einen zeitschrittbasierten Pulsgenerator ersetzt werden. Bei der Transformation der Parameter *Periode* und *Pulsbreite* musste lediglich die verwendete Abtastrate von 0.01 berücksichtigt werden: Die Periode wurde von 1/60s auf 2 samples und die Periodenbreite von 50% auf 1 sample geändert. Der im Subsystem

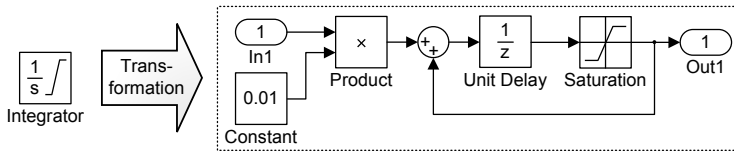


Abb. 47: Transformation eines *Integrator* Blocks in ein semantisch äquivalentes Konstrukt aus Simulink Standardblöcken (notwendig für die Anwendbarkeit von Reactis Tester).

Interior Dynamics verwendete *Integrator* Block wird ebenfalls nicht unterstützt und musste mittels eines *Unit Delay* Blocks, wie in Abb. 47 dargestellt, modelliert werden.

MODELLSPEZIFIKA: TO2

Das Modell reagiert auf die Stellung der Drosselklappe (Eingabe E1). Diese ist im ursprünglichen Modell mit Hilfe eines *Repeating Table* Blocks definiert. Dieser wurde durch eine reellwertige Systemeingabe ersetzt. Es werden Drosselklappenstellungen zwischen 3° und 90° erwartet.

System-
eingaben

Das ursprüngliche Modell verwendet einen nicht unterstützten *State Space* Block (im Subsystem */engine gas dynamics/Mixing & Combustion*). Dieses musste, wie in Abb. 48 dargestellt, mit Hilfe eines *Unit Delay* Blocks und diversen *Sum* und *Product* Blöcken für diesen speziellen Fall nachmodelliert werden. Ebenso wird ein nicht unterstützter *Integrator* Block verwendet (in Subsystem */engine gas dynamics/Throttle & Manifold/Intake Manifold*), welcher ebenfalls wie für TO1 ersetzt werden musste. Darüber hinaus verwendete das enthaltene SF Diagramm die impliziten Ereignisse „exit“ und „enter“, welche ebenfalls nicht unterstützt werden. Diese mussten entsprechend der Semantik wie folgt ersetzt werden: Das notwendige Transitionseignis „exit(Multifail)“ musste durch die Transitionsbedingung „[in(FL1)]“ und „enter(MultiFail)“ durch „[in(FL2)]“ ersetzt werden.

Anpas-
sungen für
Reactis

MODELLSPEZIFIKA: TO3

Das Modell reagiert auf zwei Eingaben: Die Pedalstellungen des Brems- (E1) und des Beschleunigungspedals (E2). Diese beiden

System-
eingaben

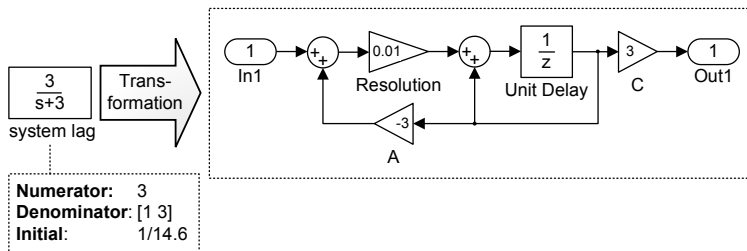


Abb. 48: Transformation des *State Space* Blocks „system lag“ von Testobjekt TO2 in ein semantisch äquivalentes Konstrukt aus Simulink Standardblöcken (notwendig für die Anwendbarkeit von Reactis Tester).

Signale entstammen im ursprünglichen Modell einem *Signal Builder* Block. Dieser wurde durch zwei reellwertige Systemeingaben ersetzt. In beiden Fällen werden Pedalstellungen im Bereich von 0% bis 100% erwartet.

Anpas-
sungen für
Reactis

Das ursprüngliche Modell verwendet insgesamt zwei von Reactis Tester nicht unterstützte Integratoren (in den Subsystemen */Engine* und */Vehicle*). Diese mussten wie im Fall von TO1 ersetzt werden. Darüber hinaus wurde im Subsystem */Vehicle* ein Funktionsblock (Fcn) verwendet, welcher eine quadratische Gleichung realisiert und somit ebenfalls nicht unterstützt wird. Dieser musste, wie in Abb. 49 dargestellt, durch Modellierung der Funktion mit Hilfe von *Sum* und *Product* Blöcken ersetzt werden.

MODELLSPEZIFIKA: TO4

System-
eingaben

Das Modell besitzt insgesamt sieben Eingaben. Zum einen benötigt das Modell die Stellungen des Wischerschalters (E1) und des Wischerhebels (E2) – gegeben als ganzzahlige Werte im Intervall [1, 3] bzw. [0, 2]. Zum anderen werden die Fahrzeuggeschwindigkeit (E3) und die Regenintensität (E4) benötigt, bei denen es sich jeweils um reellwertige Signale im Bereich von 0 bis 100 handelt. Ferner werden boolesche Signale der vorderen Fahrzeugtüren (E5 und E6) und der Motorhaube (E7) berücksichtigt, welche den jeweiligen Öffnungszustand aufzeigen.

Anpas-
sungen für
Reactis

Das ursprüngliche Modell verwendet einen von Reactis Tester nicht vollständig unterstützten *Discrete-Time Integrator* Block. Der vorhandene *State Port* dieses Blocks wird von Reactis Tester nicht

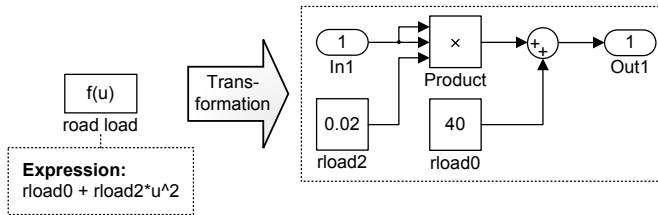


Abb. 49: Transformation des *Fcn* Blocks „road load“ von Testobjekt TO3 in ein semantisch äquivalentes Konstrukt aus Simulink Standardblöcken (notwendig für die Anwendbarkeit von Reactis Tester).

unterstützt. An Stelle der State Port Ausgabe des Blocks wird aus diesem Grund die mit Hilfe eines *Unit Delay* Blocks verzögerte Ausgabe des Integrator Blocks verwendet. Darüber hinaus waren keine Anpassungen nötig.

5.2.3 VERSUCHSAUFBAU

Zur Untersuchung der aufgestellten Propositionen kommen in der Fallstudie unterschiedliche Werkzeuge und Konfigurationen zum Einsatz:

- A. *SBGen-SL* wird zur Generierung von Teststimuli für die ausgewählten Testobjekte verwendet.
- B. *SBGen-SL* wird ohne Verwendung eines Suchverfahrens zur Generierung von Teststimuli für die ausgewählten Testobjekte verwendet (*Zufallstest*).
- C. Das Werkzeug *Reactis Tester* wird zur Generierung von Teststimuli für die ausgewählten Testobjekte verwendet.

Die jeweiligen Konfigurationen von *SBGen-SL* und *Reactis Tester* werden in den folgenden Abschnitten im Detail beschrieben.

Die Experimente werden auf einem PC mit Intel Core 2 Duo Prozessor mit einer Taktfrequenz von 2.40 GHz und 3 GB Arbeitsspeicher durchgeführt. Als Betriebssystem kommt Windows XP Professional 2002 mit Service Pack 3 zum Einsatz.

KONFIGURATION VON SBGEN-SL

Die Architektur von SBGen-SL wurde in Abschnitt 5.1 beschrieben. Dieser Abschnitt geht auf die Konfiguration des verwendeten Suchverfahrens ein.

*Parameter-
optimierung*

Basierend auf den Ergebnissen der experimentellen Untersuchung zum Vergleich der Effektivität der in dieser Arbeit entwickelten Signalgenerierungsverfahren (vgl. Absatz 3.1.4 auf Seite 84) wird der LvSb-Ansatz als zugrundeliegendes Signalgenerierungsverfahren verwendet. Um eine optimale Konfiguration der von diesem Verfahren verwendeten evolutionären Algorithmen zu bestimmen, wurde eine experimentelle Untersuchung durchgeführt. Untersucht wurde die durchschnittliche Approximationsleistung des Verfahrens bei Variation der für die Optimierungsleistung einflussreichsten Parameter. Approximationsziel war das Beispielsignal (1) (vgl. Abb. 27a auf Seite 86). Die initiale Konfiguration des Suchverfahrens entsprach der in Tab. 2 auf Seite 83 angegebenen. Die untersuchten Parameter wurden nacheinander in einem für sie typischen Intervall variiert. Zur Sicherstellung der statistischen Aussagekraft wurden für jede Variante 50 Optimierungen durchgeführt und die Ergebnisse anschließend gemittelt. Das Ergebnis dieser Untersuchung ist in Abb. A.1 für jeden einzelnen Parameter illustriert. Die basierend auf dieser Untersuchung bestimmte optimale Konfiguration von SBGen-SL ist angegeben in Tab. 10 und soll für die Fallstudie verwendet werden. Als absolutes Abbruchkriterium der Suche wurden 200 Generationen bestimmt. Bei der gewählten Populationsgröße führt dies zu maximal 5000 Ausführungen des jeweiligen Testobjekts. Hinreichendes Abbruchkriterium ist zusätzlich das Erreichen des optimalen Fitnesswertes 0.0.

Ablauf

Für jedes Überdeckungsziel werden zur Gewährleistung der statistischen Aussagekraft der Ergebnisse 50 separate und voneinander unabhängige Optimierungsläufe durchgeführt. Alle für den Vergleich relevanten Messgrößen werden gemittelt.

KONFIGURATION VON REACTIS TESTER

Version

Das Testdatengenerierungswerkzeug Reactis Tester wurde in Version V2009.2 verwendet. Stünden initiale Testdaten zur Verfügung, so könnten diese in der „Preload Phase“ spezifiziert werden. Für diese Fallstudie ist dies nicht der Fall.

Parameter		Wert
Initialisierung	Individuen	25
	Gene	$\in [60, 200]$
Selektion	Variante	SUS, Turnier ($p \in [0.1, 0.5]$, Turniergröße = 0.45)
Rekombination	Variante	Homolog, Zwei-Punkt
	P_R	0.6
	Generationslücke	1.0
Mutation	Variante	Reduzierend ($p = 0.1$), Mehrpunkt ($p = 0.8$), Erweiternd ($p = 0.1$)
	P_{M_i}	1.0
	P_{M_c}	0.9
	P_{M_g}	0.05
	Schritt	0.1
Wiedereinfügen	Variante	Elitär
	Rate	0.9
Abbruch	Generationen	200
	Fitness	≤ 0

Tab. 10: Konfiguration von *SBGen-SL* für die experimentelle Fallstudie.

Der Testdatengenerierungsprozess ist mit Hilfe von insgesamt drei Parametern konfigurierbar. In der ersten Phase, der *Random Phase*, handelt es sich zum einen um die Anzahl der zu erstellenden Tests und zum anderen um die maximale Länge (Anzahl der Zeitschritte) dieser Tests. Diese beiden Parameter wurden mit den vorkonfigurierten Standardparametern belegt: 5 zu erstellende Tests mit jeweils maximal 100 Zeitschritten.

*Random
Phase*

In der zweiten Phase, der *Targeted Phase*, kann lediglich eine Obergrenze für die maximal zur Verfügung stehenden Zeitschritte angegeben werden. Reactis Tester sucht so lange nach überdeckungssteigernden Testdaten, bis diese Anzahl der Zeitschritte ausgeschöpft ist. Diese obere Grenze liegt in der Standardparame-

*Targeted
Phase*

Parameter		Wert
Random Phase	Anzahl der Tests	5
	Max. Testlänge	100
Targeted Phase	Max. kummulative Testlänge	200.000
Überdeckung		Conditions

Tab. 11: Konfiguration von *Reactis Tester* für die experimentelle Fallstudie.

trierung bei 20.000 Zeitschritten und wurde für die Fallstudie auf 200.000 angehoben. Eine weitere Anhebung dieser Grenze hatte in vorangehenden Experimenten keine höhere Überdeckung zur Folge.

Ablauf

Als zu erreichendes Überdeckungskriterium wurde die Bedingungsüberdeckung (Conditions) gewählt. Tab. 11 fasst die gewählte Konfiguration des *Reactis Testers* für die Testdatengenerierung zusammen. Da die gesonderte Überdeckung einzelner Überdeckungsziele von *Reactis Tester* nicht unterstützt wird, werden für jedes Testobjekt insgesamt 50 separate und voneinander unabhängige Durchläufe für das gesamte Testobjekt durchgeführt. Das Erreichen der einzelnen Überdeckungsziele wird jeweils anschließend untersucht. Alle für den Vergleich relevanten Messgrößen (vgl. Abschnitt 5.2.4 auf der nächsten Seite) werden über die 50 Läufe gemittelt.

TESTSPEZIFIKATION

Tab. 12 beschreibt die in der Fallstudie angewandte Testspezifikation für die zuvor beschriebenen Testobjekte. In der ersten Spalte sind die vier Testobjekte aufgeführt. Die zweite Spalte beschreibt die Signalspezifikation für jedes einzelne Testobjekt. Die Signalspezifikation von Testobjekt TO1 beispielsweise enthält die beiden Signale s_1 und s_2 , welche jeweils reellwertige Zahlen im Intervall von -99 bis 100 enthalten können. Signal s_1 ist für Systemeingabe E1 (User Setpoint) und Signal s_2 für Systemeingabe E2 (External Temperature) vorgesehen. Die dritte Spalte der Tabelle beschreibt die jeweilige Constraint Spezifikation. Diese ist lediglich für das reale Testobjekt TO4 gegeben. Sie enthält drei Constraints für die drei Sensoren, welche jeweils nur dann aktiv sein sollen, wenn die Fahrzeuggeschwindigkeit kleiner oder gleich $5 \frac{\text{km}}{\text{h}}$ beträgt. Diese

TO	Signalspezifikation	Constraint Spezifikation
TO1	$S = \bigcap_{n=1}^2 s_n,$ $s_1 : [-99, 100] \text{ (E1)},$ $s_2 : [-99, 100] \text{ (E2)}.$	$C = \emptyset$
TO2	$S = s_1,$ $s_1 : [3, 90] \text{ (E1)}.$	$C = \emptyset$
TO3	$S = \bigcap_{n=1}^2 s_n,$ $s_1 : [0, 100] \text{ (E1)},$ $s_2 : [0, 100] \text{ (E2)}.$	$C = \emptyset$
TO4	$S = \bigcap_{n=1}^7 s_n,$ $s_1 : \{1, 2, 3\} \text{ (E1)},$ $s_2 : \{0, 1, 2\} \text{ (E2)},$ $s_3 : [0, 100] \text{ (E3)},$ $s_4 : [0, 100] \text{ (E4)},$ $s_5 : \{0, 1\} \text{ (E5)},$ $s_6 : \{0, 1\} \text{ (E6)},$ $s_7 : \{0, 1\} \text{ (E7)}.$	$C = \bigcap_{n=1}^3 c_n,$ $c_1 : (s_3 > 5) \rightarrow (s_5 == 0),$ $c_2 : (s_3 > 5) \rightarrow (s_6 == 0),$ $c_3 : (s_3 > 5) \rightarrow (s_7 == 0).$

Tab. 12: Testspezifikation für die in der Fallstudie verwendeten Testobjekte, jeweils bestehend aus Signal- und Constraint Spezifikation.

Constraints sind in diesem Fall mit Hilfe der Implikation realisiert worden (vgl. Abschnitt 5.2.2 auf Seite 154).

5.2.4 GEPLANTE DATENERHEBUNG

Die Fallstudie wird durchgeführt unter gezielter Manipulation der unabhängigen experimentellen Variablen und gleichzeitiger Untersuchung der damit einhergehenden Auswirkungen auf die abhängigen experimentellen Variablen. Die abhängigen Variablen wurden so gewählt, dass Aussagen bezüglich der in Abschnitt 5.2.1 aufgeführten Propositionen getroffen werden können.

Die folgenden drei Abschnitte beschreiben sowohl die unabhängigen und abhängigen experimentellen Variablen als auch die geplante Manipulation der unabhängigen Variablen.

UNABHÄNGIGE VARIABLEN

- u1 Als SL Modelle gegebene Testobjekte mitsamt der sich gemäß der Bedingungsüberdeckung ergebenden Liste zu erreichender Überdeckungsziele.
- u2 Verwendetes Testdatengenerierungsverfahren als Untersuchungsgrundlage. Es kommen die folgenden drei Verfahren zur Anwendung:
 - Zufallstest,
 - SBGen-SL,
 - Reactis Tester.

ABHÄNGIGE VARIABLEN

- A1 Die durchschnittlich erreichte strukturelle Überdeckung des gesamten Testobjekts.
- A2 Die durchschnittliche Erreichung/Überdeckung eines jeden strukturellen Überdeckungsziels.
- A3 Der durchschnittliche zeitliche Aufwand der Testdatengenerierung.
- A4 Die Erfüllung der an die generierten Testdaten gestellten Bedingungen (Signal Constraints).

MANIPULATION DER UNABHÄNGIGEN VARIABLEN

Für jedes ausgewählte Testobjekt (unabhängige Variable A1) werden die Experimente für jedes zu untersuchende Testdatengenerierungsverfahren (unabhängige Variable A2) separat und unabhängig voneinander durchgeführt.

Durch die Auswahl der vier Testobjekte (vgl. Abschnitt 5.2.2 auf Seite 154) werden demnach insgesamt zwölf Experimente durchgeführt.

5.2.5 KRITERIEN ZUR ERGEBNISINTERPRETATION

Zur Beurteilung von Proposition P₁ wird die erreichte strukturelle Überdeckung des jeweiligen Testobjekts (abhängige Variable A₁) als direkte Messgröße für die Effektivität verwendet. Um diesbezüglich einen detaillierteren Einblick zu erhalten, wird zusätzlich die Überdeckung der einzelnen strukturellen Überdeckungsziele betrachtet (abhängige Variable A₂). P₁

Um die Effizienz der Testdatengenerierungsverfahren zur Beurteilung von Proposition P₂ zu untersuchen, wird der durchschnittliche zeitliche Aufwand (abhängige Variable A₃) der einzelnen Verfahren vergleichend gegenübergestellt. Die Experimente werden für alle Verfahren unter gleichen Bedingungen durchgeführt, so dass die Vergleichbarkeit gewährleistet ist. P₂

Zur Beurteilung von Proposition P₃ wird die Plausibilitätsuntersuchung der generierten Testdaten durch eine Berechnung der Abweichung der Testdaten von den spezifizierten Signal Constraints durchgeführt (abhängige Variable A₄). Diese Abweichung wird als direktes Maß für die Plausibilität der Testdaten verwendet. Da lediglich Testobjekt TO₄ Constraints besitzt, kann Proposition P₃ nur für dieses untersucht werden. P₃

5.2.6 BEDROHUNG DER FALLSTUDIENVALIDITÄT

Bei der Auswertung der Fallstudienenergebnisse müssen die möglichen Bedrohungen der Validität der Fallstudie beachtet werden. Dieser Abschnitt geht auf unterschiedliche Bedrohungen sowohl für die Konstruktvalidität, als auch für die interne und die externe Validität der Fallstudie ein.

KONSTRUKTVALIDITÄT

Bedrohungen der Konstruktvalidität beziehen sich auf das Konstrukt der Fallstudie und die Zulässigkeit der zugrundeliegenden Operationalisierung. Kann durch die gewählten experimentellen Variablen die untersuchte Hypothese angemessen beurteilt werden?

Mögliche Bedrohungen der Konstruktvalidität:

- Der zeitliche Aufwand der Testdatengenerierung (abhängige Variable A₃) unterscheidet sich je nach Rechnertyp und -auslastung. Um diese Bedrohung zu minimieren, werden alle Experimente auf dem selben Rechner und unter gleichen Bedingungen ausgeführt.

INTERNE VALIDITÄT

Die interne Validität befasst sich mit der Interpretierbarkeit der Ergebnisse. Sie bemisst, inwieweit die Veränderungen der abhängigen Variablen aus den Änderungen der unabhängigen Variablen resultieren. Die interne Validität beschreibt, in welchem Ausmaß die Fallstudie durch die Existenz von Störvariablen und anderen unerwarteten Einflussfaktoren beeinflusst werden kann.

Mögliche Bedrohungen der internen Validität:

- Instabilitäten bzw. Fehler in der prototypischen Implementierung von SBGen-SL sind unwahrscheinlich, können jedoch nicht ausgeschlossen werden.
- Der für evolutionäre Algorithmen immanente Nichtdeterminismus. Diese Bedrohung wird durch das mehrfache Ausführen der Fallstudien und das Mitteln der Ergebnisse weitestgehend entschärft.

EXTERNE VALIDITÄT

Bedrohungen der externen Validität der Fallstudie beeinflussen die Verallgemeinerbarkeit der gezogenen Schlussfolgerungen.

Mögliche Bedrohungen der externen Validität:

- Die Repräsentativität der ausgewählten Testobjekte. Alle in dieser Arbeit verwendeten Testobjekte entstammen dem Automobilbereich, was zur Folge haben könnte, dass die Ergebnisse der Fallstudie lediglich für diese industrielle Domäne gültig sind. Diese Spezialisierung wurde aufgrund der besonderen Relevanz von SL Modellen in der Automobilindustrie vorgenommen. Das Wiederholen der Fallstudie in anderen Domänen würde die notwendigen Informationen zur besseren Verallgemeinerung der Ergebnisse liefern.

- Die generierten Teststimuli könnten nicht real genug und die erzielten Ergebnisse daher nicht für reale Systeme verallgemeinerbar sein. Die Realitätsnähe der generierten Signale wird durch die gezielte Erfüllung der gegebenen Signal Constraints gewährleistet.
- Die Anzahl der Wiederholungen der Experimente könnte nicht ausreichen, so dass dies zu einer geringen statistischen Bedeutung der Ergebnisse führen könnte. Eine Erhöhung der Wiederholungsanzahl ist jedoch aufgrund des hohen zeitlichen Aufwands nicht praktikabel. Es werden Tests zur Analyse der statistischen Aussagekraft der Ergebnisse durchgeführt.

5.3 DURCHFÜHRUNG DER FALLSTUDIE

Die Fallstudie wurde wie in Abschnitt 5.2.3 beschrieben durchgeführt. Die Auswirkung der in Abschnitt 5.2.4 beschriebenen Manipulation der unabhängigen auf die abhängigen experimentellen Variablen wurde untersucht. In Abschnitt 5.3.1 werden die erhaltenen Messwerte vorgestellt und anschließend in Abschnitt 5.3.2 bewertet.

5.3.1 DATENERHEBUNG

Zunächst mussten die Testobjekte bezüglich der Erreichbarkeit der enthaltenen Überdeckungsziele untersucht werden. Verschiedene konstant definierte Modellparameter können dazu führen, dass einzelne Überdeckungsziele unabhängig von der Stimulation des Modells nicht erreicht werden können. Das Ergebnis dieser Untersuchung ist in Abb. 50 illustriert. Demnach sind 26% der Überdeckungsziele von Testobjekt TO₁ nicht erreichbar, während für Testobjekt TO₃ alle enthaltenen Überdeckungsziele erreichbar sind. Bei Testobjekt TO₂ bzw. Testobjekt TO₄ sind hingegen 47% bzw. 39% der Überdeckungsziele nicht erreichbar. Unerreichbare Überdeckungsziele werden in der Fallstudie nicht berücksichtigt. Zum einen kann so der ansonsten notwendige Rechenaufwand eingespart werden. Zum anderen bezieht sich die hundertprozentige Überdeckung des jeweiligen Testobjekts so nur auf die

*Erreichbare
Überde-
ckungsziele*

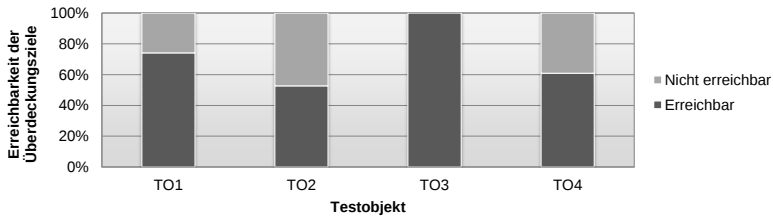


Abb. 50: Prozentuale Erreichbarkeit der Überdeckungsziele der vier untersuchten Testobjekte.

tatsächlich erreichbaren Überdeckungsziele. Abschnitt A.2 auf Seite 190 führt die einzelnen strukturellen Überdeckungskriterien der ausgewählten Testobjekte mitsamt derer Erreichbarkeit auf.

Die folgenden Abschnitte gehen auf die Ergebnisse der einzelnen Testdatengenerierungsverfahren ein. Für jedes dieser Verfahren wird die erreichte Überdeckung der Testobjekte (abhängige Variable A1), die durchschnittliche Ausführungszeit (A3) und die durchschnittliche Constraintbefriedigung der generierten Testdaten (A4) vorgestellt. Die Ergebnisse für die einzelnen Überdeckungsziele jedes Testobjekts (A2) sind in Abschnitt A.3 auf Seite 196 aufgeführt.

ZUFALLSTEST

Tabellenstruktur

Die Ergebnisse der Fallstudie für die Zufallssuche sind in Tab. 13 dargestellt. Die Spalte *TO* führt die einzelnen Testobjekte auf. Die Spalte *Überdeckung* gibt die für das jeweilige Testobjekt durchschnittlich erreichte strukturelle Überdeckung in Prozent an. Die Spalte *Ausführungszeit* beschreibt die durchschnittlich benötigte Zeit (in Minuten) zur Überdeckung des Testobjekts bzw. falls keine vollständige Überdeckung erreicht wurde, zur Erreichung des garantierten Abbruchkriteriums. Die Spalte *Constraints* hingegen stellt die Erfüllung der Signal Constraints in Prozent dar (gilt nur für TO4). Die angegebenen Werte sind für die durchgeführten 50 Wiederholungen gemittelt, die jeweils zugehörige Standardabweichung ist in Klammern angegeben.

Ergebnis

Die Ergebnisse verdeutlichen, dass der Zufallstest bei den untersuchten Testobjekten bereits zu einer relativ hohen strukturellen Überdeckung führte. Es konnte eine Überdeckung von gut 75% bis gut 98% erreicht werden. Gemittelt über alle Testobjekte wurde

TO	Überdeckung	Ausführungszeit	Constraints
TO1	96,09 % (0,68 %)	100,67 m (12,44 m)	–
TO2	86,79 % (12,43 %)	351,63 m (5,81 m)	–
TO3	98,17 % (6,35 %)	27,13 m (3,32 m)	–
TO4	75,54 % (5,21 %)	2.092,43 m (207,55 m)	0,08 % (0,03 %)
∅	89,15 % (6,17 %)	642,97 m (57,28 m)	0,08 % (0,03 %)

Tab. 13: Ergebnisse der Fallstudie für den *Zufallstest*. Die Werte sind über 50 Ausführungen gemittelt; die jeweils zugehörige Standardabweichung ist in Klammern angegeben.

eine Überdeckung von gut 89% erreicht. Diese Angaben beziehen sich auf die jeweils erreichbaren Überdeckungsziele. Aufgrund des ausgeprägten Nichtdeterminismus des Zufallstests ergeben sich erwartungsgemäß verhältnismäßig hohe Standardabweichungen. Die durchschnittlichen Ausführungszeiten beliefen sich auf den Bereich von 27,13 Minuten (TO3) bis knapp 35 Stunden (TO4). Die für Testobjekt TO4 spezifizierten Signal Constraints wurden im Schnitt lediglich zu 0,083% (Standardabweichung: 0,03%) erfüllt. Dieses Ergebnis ist nicht verwunderlich, da keinerlei zielgerichtete Suche nach entsprechenden Testdaten stattfand.

Der Grund für die vergleichsweise gute Leistung der Zufallssuche liegt in der Trivialität eines Großteils der Überdeckungsziele. Über alle Testobjekte gemittelt, handelt es sich bei 79,5% aller erreichbaren Überdeckungsziele um derartige triviale Überdeckungsziele. Diese wurden in jedem Lauf der durchgeführten Experimente von der Zufallssuche überdeckt. Dieses Ergebnis deckt sich mit den Ergebnissen von Harman und McMinn, die ebenfalls zeigten, dass die Zufallssuche bereits einen großen Teil eines Testobjekts überdeckt und lediglich für wenige nicht-triviale Überdeckungsziele ein differenziertes Testdatengenerierungsverfahren von Nöten ist [58].

Interpretation

REACTIS TESTER

Die Ergebnisse der Fallstudie für Reactis Tester sind in Tab. 14 dargestellt. Die Struktur der Tabelle entspricht jener von Tab. 13.

Mit Reactis Tester konnte eine Überdeckung von gut 70% bis gut

Ergebnis

TO	Überdeckung	Ausführungszeit	Constraints
TO1	97,12 % (0,69 %)	39,97 m (1,25 m)	–
TO2	70,37 % (0,00 %)	44,87 m (1,43 m)	–
TO3	95,67 % (10,44 %)	35,37 m (1,16 m)	–
TO4	88,27 % (1,92 %)	61,42 m (1,41 m)	0,08 % (0,02 %)
⊙	87,53 % (3,26 %)	45,41 m (1,31 m)	0,08 % (0,02 %)

Tab. 14: Ergebnisse der Fallstudie für *Reactis Tester*. Die Werte sind über 50 Ausführungen gemittelt; die jeweils zugehörige Standardabweichung ist in Klammern angegeben.

97% erreicht werden; gemittelt über alle Testobjekte wurde für alle erreichbaren Überdeckungsziele eine Überdeckung von knapp 88% erreicht. Für die Testobjekte TO1, TO2 und TO4 erzielte Reactis Tester im Vergleich zum Zufallstest eine höhere Überdeckung. Insbesondere für das reale Testobjekt TO4 ist der Unterschied deutlich erkennbar. Die einzige Ausnahme stellt Testobjekt TO3 dar, für welches eine 2,50% geringere Überdeckung erzielt wurde. Die durchschnittliche Ausführungszeit war mit 45,41 Minuten im Vergleich deutlich kürzer. Die für Testobjekt TO4 spezifizierten Signal Constraints wurden im Schnitt ebenfalls zu 0,08% (Standardabweichung: 0,02%) erfüllt. Ebenso wie bei der Zufallssuche wurden die Constraints im Testdatengenerierungsprozess nicht berücksichtigt, wodurch sich dieses Ergebnis erklärt.

Komplexe Prädikate

Eine Analyse der Ergebnisse erlaubt die folgende Begründung für die teilweise nicht erreichten Überdeckungsziele: Das Erreichen vieler Überdeckungsziele ist abhängig von komplexen Prädikaten. Bei TO1 muss für Überdeckungsziel *tg_023* die absolute Temperaturdifferenz zwischen Innenraum und Umwelt größer als 203°K betragen. Die Innenraumtemperatur wird über komplexe nicht-lineare Berechnungen bestimmt. Es muss darüber hinaus von einem Fehler im Modell ausgegangen werden, da diese Temperaturdifferenz bei den gegebenen Schranken für die beiden Systemeingaben physikalisch nicht möglich sein dürfte. Ein weiteres Beispiel ist Überdeckungsziel *tg_0326* von Testobjekt TO2: Zu dessen Erfüllung werden Bedingungen an den Druck im Ansaugkrümmer gestellt. Dieser Druck wiederum berechnet sich ebenfalls über komplexe nicht-lineare Gleichungen. In TO4 sind demgegenüber nur wenige – als solches – komplexe Bedingungen enthalten. Aus der Größe des Modells und der starken internen Vernetzung jedoch,

TO	Überdeckung	Ausführungszeit	Constraints
TO1	99,81 % (0,27 %)	75,17 m (5,58 m)	–
TO2	96,29 % (0,00 %)	174,97 m (2,72 m)	–
TO3	99,67 % (1,15 %)	16,67 m (2,25 m)	–
TO4	94,42 % (2,74 %)	991,60 m (46,35 m)	98,08 % (1,69 %)
⊘	97,55 % (1,04 %)	314,60 m (14,23 m)	98,08 % (1,69 %)

Tab. 15: Ergebnisse der Fallstudie für *SBGen-SL*. Die Werte sind über 50 Ausführungen gemittelt; die jeweils zugehörige Standardabweichung ist in Klammern angegeben.

ergibt sich auch hier eine Vielzahl komplexer Prädikate. Wann immer ein Überdeckungsziel von der Erfüllung komplexer Prädikate abhängig ist, konnte eine Verringerung der durchschnittlichen Effektivität von Reactis Tester beobachtet werden. Zur Erreichung einiger Überdeckungsziele musste eine bestimmte Variable auf einen spezifischen Wert bzw. auf einen Wert innerhalb eines engen Intervalls gebracht werden. Bei TO1 beispielsweise muss sich die absolute Temperaturdifferenz für die Überdeckungsziele *tg_0322*, *tg_0332*, *tg_0342* und *tg_0352* innerhalb eines relativ engen Intervalls befinden. Die Größe des jeweiligen Wertebereiches und das Ausmaß der notwendigen Einschränkung haben einen direkten Einfluss auf die durchschnittliche Effektivität von Reactis Tester. Einige Überdeckungsziele stellten Anforderungen an die minimale Länge der generierten Signale. Bei TO2 beispielsweise hängen viele Überdeckungsziele davon ab, dass sich das System länger als 4,8 Sekunden in Zustand *O2_warmup* befindet. Wann immer dies der Fall war, führte dies zu deutlichen Effektivitätseinschränkungen von Reactis Tester. Grund dafür ist die Generierung von Signalverläufen mit einer überwiegend besonders geringen Anzahl von Zeitschritten. Wie in Abb. A.3 dargestellt, konnte aus diesem Grund sogar die Zufallssuche im Vergleich zu Reactis Tester ein bzgl. der Effektivität besseres Ergebnis erzielen; Die Zufallssuche erzeugt Signale mit zufälliger Länge.

*Spezifische
Werte*

*Minimale
Signallänge*

SBGEN-SL

Die Ergebnisse der Fallstudie für *SBGen-SL* sind in Tab. 15 dargestellt. Die Struktur der Tabelle entspricht ebenfalls jener von Tab. 13.

Ergebnis SBGen-SL erreichte mit einer über alle Testobjekte gemittelten Überdeckung von 97,55% die höchste Überdeckung. Diese reicht von 94,42% für Testobjekt TO4 bis 99,81% für Testobjekt TO1. Während sowohl die Zufallssuche als auch Reactis Tester für Testobjekt TO2 lediglich eine Überdeckung von 70,37% erreichen konnten, erreichte SBGen-SL eine Überdeckung von 96,29%. Die durchschnittliche Ausführungszeit für die untersuchten Testobjekte ist vergleichbar mit jener der Zufallssuche; sie reichte von knapp 17 Minuten (TO3) bis gut 16 Stunden (TO4). Der größte Unterschied besteht erwartungsgemäß in der erreichten Erfüllung der spezifizierten Signal Constraints für Testobjekt TO4. Diese wurden im Schnitt zu 98,08% (Standardabweichung: 1,69%) erreicht.

Starke Zustandsbehaftung Eine Analyse der Ergebnisse erlaubt die folgende Begründung für die nicht erreichten Überdeckungsziele: Zum Erreichen der drei nicht von SBGen-SL erreichten Überdeckungsziele muss jeweils eine spezifische Kombination aus unterschiedlichen Zuständen eingenommen werden. Dazu müssen diverse Zustände verschiedener SF Diagramme durchlaufen werden. Für die einzelnen notwendigen Zustandswechsel muss wiederum eine Vielzahl von Variablen eine spezifische Wertebelegung aufweisen und bestimmte SF Diagramme müssen sich in spezifischen Zuständen befinden. Erschwerend kommt hinzu, dass die zum Betreten eines Zustands notwendige Bedingung für das Betreten des Folgezustands teilweise die gegenteilige Aussage aufweisen muss. Eine bestimmte Variablenbelegung wird somit in unterschiedlichen Zeitschritten – in Abhängigkeit des jeweiligen Systemzustands – unterschiedlich bewertet. Dies stellt eine besondere Herausforderung für das verwendete Suchverfahren dar, insbesondere auch weil die Bewertungen der einzelnen Zeitschritte zu einer Gesamtbewertung zusammengefasst werden.

5.3.2 BEWERTUNG

Die mit vier SL Testobjekten durchgeführte experimentelle Fallstudie konnte die Effektivität des suchbasierten Strukturtests für SL Modelle (implementierten im Testdatengenerator SBGen-SL) nachgewiesen werden. Sowohl im Vergleich zur Zufallssuche als auch im Vergleich zum kommerziellen Werkzeug Reactis Tester, erreichte SBGen-SL eine höhere durchschnittliche Überdeckung

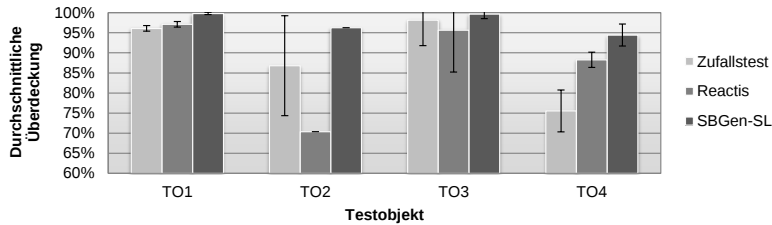


Abb. 51: Vergleich der Testdatengenerierungsverfahren bzgl. der erreichten strukturellen Überdeckung der vier Testobjekte (gemittelt über jeweils 50 Experimente).

aller Testobjekte. Abb. 51 stellt die jeweils erzielte Überdeckung für die einzelnen Testobjekte vergleichend gegenüber.

Im folgenden werden die in Abschnitt 5.2.1 bestimmten Propositionen basierend auf den vorgestellten Ergebnissen bewertet.

BEWERTUNG VON PROPOSITION P1

Für alle Testobjekte gemittelt ergibt sich für jedes Testdatengenerierungsverfahren die in Abb. 52a illustrierte durchschnittliche Überdeckung. Es ist deutlich zu erkennen, dass SBGen-SL im Vergleich zur Zufallssuche und zu Reactis Tester eine signifikant höhere Effektivität aufweist. Gemessen an der Zufallssuche wurde im Schnitt eine um 7,4% höhere Überdeckung erreicht. Im Vergleich mit Reactis Tester fällt der Unterschied mit knapp 10% noch deutlicher aus. Die Unterlegenheit von Reactis Tester gegenüber des Zufallstests ist auf die deutlichen Schwächen bei Testobjekt TO2 zurückzuführen.

Proposition P1 kann somit bestätigt werden: SBGen-SL weist im Vergleich zu existierenden Testdatengenerierungsverfahren (repräsentiert durch die Zufallssuche und einen kommerziellen Ansatz) eine höhere Effektivität bei der Testdatengenerierung auf. Es erreichte in jedem Fall eine höhere strukturelle Überdeckung der Testobjekte.

BEWERTUNG VON PROPOSITION P2

Die Effizienz der Verfahren – gemessen in Form der durchschnittlichen Ausführungszeit – ist in Abb. 52b vergleichend gegen-

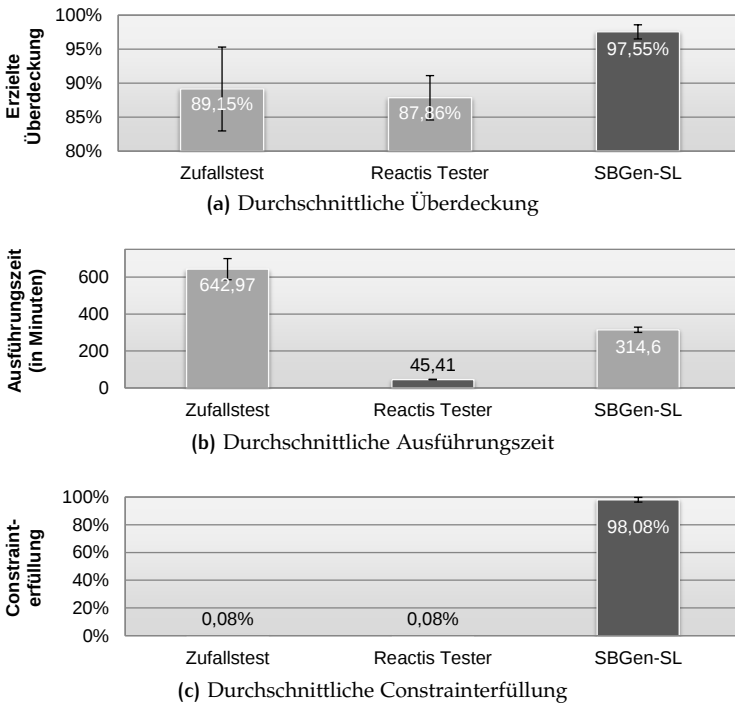


Abb. 52: Ergebnisse der Fallstudie für die untersuchten Testdatengeneratoren, gemittelt über alle Testobjekte.

übergestellt. Aufgrund der bei Reactis Tester fest parametrisierten maximalen Anzahl von Simulationsschritten innerhalb des Testdatengenerierungsprozesses ergibt sich für jedes Testobjekt eine ähnliche Ausführungszeit: Im Mittel gut 45 Minuten. Bei der Zufallssuche und bei SBGen-SL werden die Überdeckungsziele sequentiell abgearbeitet. Für während des Testdatengenerierungsprozesses zufällig erreichte Überdeckungsziele muss keine separate Suche durchgeführt werden. Dieser Nichtdeterminismus führt zu stark variierenden Ausführungszeiten: Bei der Zufallssuche belief sich diese im Schnitt auf gut 640 Minuten und bei SBGen-SL auf knapp 315 Minuten. Diese im Vergleich zu Reactis Tester deutlich höheren Ausführungszeiten sind auf die bei dynamischen Testverfahren inherente vielfache Ausführung des Testobjekts zurückzuführen.

Proposition P₂ konnte nicht bestätigt werden: SBGen-SL weist im Vergleich zu existierenden Testdatengenerierungsverfahren (repräsentiert durch die Zufallssuche und einen kommerziellen Ansatz) keine höhere Effizienz bei der Testdatengenerierung auf. Die hohen Ausführungszeiten können jedoch durch eine Parallelisierung der Testobjektausführung drastisch reduziert werden. So könnte beispielsweise jede Population von Lösungsvorschlägen auf einem Mehrkernsystem bzw. auf mehreren Rechnern parallel zur Ausführung gebracht werden. The MathWorks™, Inc. stellen entsprechende Werkzeuge für SL bereit [131]. Bei einer Populationsgröße von 20 Individuen, kann die Ausführungszeit dadurch um bis zu 95% reduziert werden. Somit könnte auch Proposition P₂ bestätigt werden.

BEWERTUNG VON PROPOSITION P₃

Die Plausibilität der generierten Testdaten – gemessen als die durchschnittliche Erfüllung gegebener Signal Constraints – ist in Abb. 52c vergleichend dargestellt. Dies wurde nur für das reale Testobjekt TO₄ untersucht. Da sowohl bei der Zufallssuche als auch bei Reactis Tester die Erfüllung von Constraints nicht im Testdatengenerierungsprozess berücksichtigt wird, ist erwartungsgemäß eine Erfüllung von 0,08% beobachtet worden. Dies entspricht in etwa der Wahrscheinlichkeit, dass ein Testdatum bei gegebenem Suchraum diese Constraints zufällig erfüllt. Demgegenüber konnte SBGen-SL beinahe in jedem Fall eine vollständige Erfüllung der Constraints erzielen. Im Schnitt wurde eine Constraintbefriedigung von 98,08% erreicht.

Proposition P₃ kann somit bestätigt werden: SBGen-SL erzeugt im Gegensatz zur Zufallssuche und zu Reactis Tester plausible Testdaten für Testobjekte mit geltenden Signal Constraints.

5.4 ZUSAMMENFASSUNG

Dieses Kapitel befasste sich mit der experimentellen Fallstudie zur Bewertung des in dieser Arbeit entwickelten Testdatengenerierungsverfahrens. Das Ziel war die Beantwortung der Forschungsfrage. Diese konnte mittels einer Untersuchung, sowohl

hinsichtlich der Effektivität und der Effizienz der Testdatengenerierung als auch hinsichtlich der Plausibilität der generierten Testdaten, beantwortet werden.

In den Experimenten wurden die Leistungen des Verfahrens – implementiert im Testdatengenerator *SBGen-SL* – mit jenen des Zufallstests und des kommerziellen Werkzeugs *Reactis Tester* vergleichend gegenübergestellt. Im Schnitt erreichte *SBGen-SL* für alle Testobjekte die höchste strukturelle Überdeckung (Effektivität). Ebenso war lediglich *SBGen-SL* in der Lage plausible Testdaten zu generieren, d.h. Testdaten, welche die gegebenen Signal Constraints erfüllen. Aufgrund der fehlenden Berücksichtigung während des Testdatengenerierungsprozesses, war dies der Zufallssuche und *Reactis Tester* nicht möglich. Diese vergleichsweise höhere Leistung hatte jedoch den Nachteil der im Vergleich zu *Reactis Tester* nachgewiesenen geringeren Effizienz der Testdatengenerierung. Über den Weg der Parallelisierung der Testobjektausführung ließe sich dieser Nachteil jedoch relativieren.

Der entwickelte Ansatz konnte erfolgreich für die Automatisierung der Aufgabe der Testdatengenerierung eingesetzt werden. Es wurden insbesondere für reale und komplexe SL/SF Modelle deutliche Vorteile gegenüber existierenden Verfahren nachgewiesen.

Teil III

ABSCHLUSS

6

ZUSAMMENFASSUNG UND AUSBLICK

*Jede Lösung eines Problems
ist ein neues Problem.*

— Johann Wolfgang von Goethe

Dieses Kapitel schließt die vorliegende Dissertation ab. Es ist in drei Unterkapitel gegliedert. Im ersten Unterkapitel werden die Ergebnisse dieser Arbeit zusammengefasst. Das zweite Unterkapitel geht auf die verbliebenen Einschränkungen und Grenzen des entwickelten Verfahrens ein. Abschließend werden im darauf folgenden Unterkapitel offene Fragestellungen und weiterführende Arbeiten diskutiert.

6.1 ERGEBNISSE DIESER ARBEIT

Das Ziel dieser Arbeit war die Entwicklung eines Ansatzes zur automatischen Generierung plausibler Testdaten für SL/SF Modelle und das Testziel der strukturellen Überdeckung. Die Notwendigkeit für einen solchen Ansatz ergab sich aus den Grenzen existierender Ansätze. Diese Grenzen verhinderten einen erfolgreichen Einsatz der Ansätze im industriellen Umfeld. Beispiele sind insbesondere eine mangelnde Anwendbarkeit für komplexe Testobjekte, der geringe Grad der erreichten strukturellen Überdeckung und die geringe Plausibilität der generierten Testdaten (vgl. Abschnitt 1.1). Zur Adressierung dieser Grenzen stellt diese Arbeit den suchbasierten Strukturtest für SL/SF Modelle vor.

*Ziel der
Arbeit*

Der suchbasierte Strukturtest ist ein dynamisches Testdatengenerierungsverfahren, dessen Stärke – auf prozeduralen Code angewandt – in einer Vielzahl von Arbeiten nachgewiesen wurde [93]. Es transformiert die Aufgabe der Testdatengenerierung in ein Optimierungsproblem und löst dieses durch Anwendung robuster Suchverfahren. Die vorliegende Arbeit macht den suchbasierten

*Suchbasiert.
Strukturtest*

Strukturtest für SL/SF Modelle anwendbar. Dabei kommt keine symbolische Ausführung und kein Constraint Solving zum Einsatz. Die ihnen inhärenten Einschränkungen haben somit keine Auswirkungen auf das vorgestellte Verfahren.

*Plausibilität
der
Testdaten*

Zur Sicherstellung der Plausibilität der generierten Testdaten wurde als erster Hauptbeitrag der Dissertation ein dedizierter Signalgenerator entwickelt und in Kapitel 3 vorgestellt. Dieser wurde für die Verwendung mit robusten Suchverfahren konzipiert und ermöglicht sowohl die Spezifizierung der Charakteristika der gewünschten Signale als auch ggf. die Definition einzuhaltender Signalconstraints. Diese in Form einer temporalen Logik spezifizierten Bedingungen werden im Testdatengenerierungsprozess berücksichtigt. Das Verfahren generiert somit lediglich Testdaten, die der vom Tester angegebenen Signalspezifikation entsprechen. Die üblicherweise anschließend durchgeführte Plausibilitätsprüfung der Testdaten wird dadurch obsolet.

*Distanz-
maße für
SL/SF*

Die dem Verfahren zugrundeliegende Suche wird basierend auf der angewandten Zielfunktion gesteuert. Grundlage für deren Berechnung sind einerseits Informationen über den während der Ausführung des Testobjekts auftretenden Datenfluss. Die notwendige Protokollierung dieses Datenflusses wird durch eine Instrumentierung des Testobjekts ermöglicht. Andererseits sind modellspezifische Distanzmaße für die Berechnung der Zielfunktion grundlegend: Unter Verwendung der protokollierten Informationen und der internen Struktur des Testobjekts quantifizieren diese Distanzmaße die Annäherung der generierten Testdaten an das jeweilige strukturelle Überdeckungsziel. Der Vorschlag einer für SL Modelle und SF Diagramme spezifischen Instrumentierung und die Entwicklung dedizierter Distanzmaße ist der zweite Hauptbeitrag dieser Dissertation (vgl. Kapitel 4).

Fallstudie

Das Verfahren ist prototypisch realisiert und durch eine Fallstudie mit insgesamt vier Testobjekten aus dem automobilen Kontext validiert worden (vgl. Kapitel 5). Diese umfangreiche experimentelle Fallstudie ist ein weiterer Beitrag dieser Dissertation. Es wurde gezeigt, dass das Verfahren die Zufallssuche bezüglich Effektivität und Effizienz der Suche übertrifft. Im Vergleich mit dem kommerziellen Testwerkzeug *Reactis Tester* wurde, sowohl in Hinblick auf die erreichte strukturelle Überdeckung (im Schnitt wurde eine knapp 10% höhere Überdeckung erreicht), als auch bezüglich der Plausibilität der generierten Testdaten, eine deutliche Überlegenheit des entwickelten Verfahrens nachgewiesen.

6.2 EINSCHRÄNKUNGEN UND GRENZEN

Der suchbasierte Strukturtest für SL/SF Modelle weist folgende Einschränkungen und Grenzen auf:

GERINGE EFFIZIENZ

Das entwickelte Verfahren zählt zu den dynamischen Testverfahren. Aufgrund der vielfachen, wiederholten Ausführung des Testobjekts hängt der zeitliche Aufwand des Verfahrens von zwei Faktoren ab: von der Ausführungszeit des Testobjekts und von der Effizienz des verwendeten Suchverfahrens. Der zuerst genannte Faktor hängt in erster Linie von der Größe und der Komplexität des Testobjekts ab. Der zweite Faktor spielt ebenfalls eine besonders wichtige Rolle: Je effizienter die Suche, desto geringer die Anzahl der notwendigen Testobjektausführungen und desto effizienter das Testdatengenerierungsverfahren. Bei besonders komplexen Modellen führt dies möglicherweise zu Skalierungsproblemen.

SEMANTISCH LOKAL ORIENTIERTE ZIELFUNKTION

Die Zielfunktion ist maßgebend für die Effektivität und Effizienz des verwendeten Suchverfahrens. Beim suchbasierten Strukturtest für SL/SF Modelle ist die Zielfunktion semantisch lokal orientiert, d.h. sie verwendet lediglich semantische Informationen aus der lokalen Umgebung des strukturellen Überdeckungsziels. Soll beispielsweise am zweiten Eingabeport eines SL *Switch* Blocks ein bestimmter Wert anliegen, so wird die Distanz der aktuellen Wertebelegung zu diesem Wert als Zielfunktionswert verwendet. Weiter entfernte, d.h. im Modell vorgelagerte Elemente, können ebenfalls einen Einfluss auf die Wertebelegung des zweiten Eingabeports ausüben, werden jedoch nicht in der Distanzberechnung berücksichtigt. Aus diesem Grund kann die optimale Steuerung der Suche nicht in jedem Fall gewährleistet werden.

BOOLESCHE BEDINGUNGEN

Wann immer die Erreichung eines Überdeckungsziels von booleschen Bedingungen abhängt, gestaltet sich die Berechnung eines

entsprechenden Distanzwertes schwierig. Da für einen booleschen Wert allein kein feingranularer Distanzwert berechnet werden kann (die Distanz ist immer 1 oder 0), muss versucht werden, hilfreiche Informationen aus den vorgelagerten SL Blöcken zu erhalten. Entspricht der fragwürdige boolesche Wert beispielsweise einem SL *Relational* Block (z.B. $x > 5$), kann dessen Eingabe zur Berechnung der gewünschten Distanz verwendet werden. Dies ist jedoch nicht immer möglich. Der suchbasierte Strukturtest verhält sich in diesen Fällen wie eine Zufallssuche, mitsamt der damit verbundenen Verringerung der Effektivität und Effizienz der Testdatengenerierung.

FEHLENDE UNTERSTÜTZUNG FÜR STATEFLOW FUNKTIONEN

Die Bedingungen einer Transition innerhalb eines SF Diagramms können auf den Ausgaben von Funktionen basieren (Call to functions). Bei diesen Funktionen handelt es sich entweder um eigene SF Diagramme, um ML Funktionen oder eingebettete C-Funktionen. Diese werden vom vorgestellten Verfahren nicht durch gesonderte Distanzberechnungen berücksichtigt.

FEHLENDE UNTERSTÜTZUNG FÜR MCDC

Das Überdeckungskriterium MC/DC wird neben dem unterstützten Überdeckungskriterium CC in der Praxis häufig gefordert. Dieses stellt jedoch wechselseitige Anforderungen an die zu generierenden Testdaten. So sollen die generierten Testdaten dazu führen, dass jede Bedingung einer Entscheidung unabhängig von allen anderen enthaltenen Bedingungen zu einem Wechsel der Entscheidung führt. Die dazu notwendige multikriterielle Suche war nicht Gegenstand dieser Arbeit. Das Überdeckungskriterium MC/DC wird daher nicht unterstützt.

6.3 WEITERFÜHRENDE ARBEITEN

Die Arbeit lässt sich in diversen Richtungen weiterführen. Zum einen kann der suchbasierte Strukturtest für SL/SF Modelle durch Behebung seiner Einschränkungen und Grenzen verbessert und dadurch dessen Anwendbarkeit in der industriellen Praxis erhöht

werden. Zum anderen könnte das Verfahren konzeptuell erweitert und theoretisch untermauert werden. Ebenso sind weitere Forschungsrichtungen und Wechselwirkungen mit anderen Bereichen der Testautomatisierung denkbar. Diese drei Strömungen werden im Folgenden angeregt.

BEHEBUNG DER EINSCHRÄNKUNGEN UND GRENZEN

Die praktische Anwendbarkeit des vorgestellten Verfahrens lässt sich durch folgende Maßnahmen signifikant erhöhen:

- Parallelisierung der Testobjektausführung
- Hybridisierung mit statischen Analysetechniken
- Optimierung des verwendeten Suchverfahrens
- Unterstützung von MC/DC

Für populationsbasierte Suchverfahren bietet sich die Parallelisierung der Testobjektausführung zur Steigerung der Effizienz an. Insbesondere wenn die Ausführungszeit des Testobjekts einen nicht vernachlässigbaren Umfang erreicht, hat die Parallelisierung besonders große Auswirkungen. Sobald vom Suchverfahren eine neue Population von Lösungsvorschlägen erstellt wurde, kann diese einem Rechnerverbund (im Idealfall Mehrkernsysteme) zur parallelen Ausführung übergeben werden. The MathWorksTM, Inc. stellen die dafür notwendigen Werkzeuge für SL bereit [131]. Es sollte daher analysiert werden, in welchem Umfang die Effizienz des suchbasierten Strukturtests für SL/SF Modelle durch Parallelisierung der Testobjektausführung unter Berücksichtigung des steigenden Kommunikationsaufwandes erhöht werden kann.

Parallelisierung

Der suchbasierte Strukturtest stößt insbesondere für boolesche Konstrukte aufgrund der fehlenden Granularität der Distanzberechnung an seine Grenzen. In diesen Fällen haben statische Verfahren deutliche Stärken. Eine Hybridisierung des suchbasierten Strukturtest mit statischen Analyseverfahren sollte aus diesem Grund zur Steigerung sowohl der Effizienz als auch der Effektivität untersucht werden. Denkbar ist die Anwendung statischer Verfahren, z.B. der symbolischen Ausführung, während der Testvorbereitung. Die dadurch aufgefundenen Testdaten könnten anschließend als Startwerte für die suchbasierte Testdatengenerierung verwendet werden.

Hybridisierung

*Optimie-
rung der
Suche*

In dieser Arbeit wurde ein längenvariabler genetischer Algorithmus als Suchverfahren eingesetzt, dessen Konfiguration durch eine experimentelle Fallstudie bestimmt wurde. Es ist zwar anzunehmen, jedoch nicht garantiert, dass diese Konfiguration für jedes Überdeckungsziel eine gute Wahl darstellt. Möglicherweise ist jeweils auch ein gänzlich anderes Suchverfahren deutlich besser geeignet. Jedes Überdeckungsziel stellt in Abhängigkeit der verwendeten Zielfunktion ein gesondertes Optimierungsproblem dar. Aufgrund des No-Free-Lunch-Theorems [151], welches sinngemäß besagt, dass kein universelles Suchverfahren für alle Optimierungsprobleme existiert, ergibt sich die Herausforderung, für jedes Überdeckungsziel das Suchverfahren mit der höchsten Erfolgswahrscheinlichkeit zu bestimmen. Steht nur ein Suchverfahren zur Verfügung, bestünde die Herausforderung in der Bestimmung einer jeweils optimalen Parametrierung. Die optimale Parameterbestimmung ist ein wichtiges Aufgabengebiet im Feld der Algorithmik. In weiterführenden Arbeiten könnten theoretische Untersuchungen bestimmen, in welchen Fällen der suchbasierte Strukturtest bzw. die verwendete Parametrierung Vorteile hat und verwendet werden sollte. Der Abgleich dedizierter Modellkomplexitätsmaße und die Ergebnisse für unterschiedliche Konfigurationen könnten einen ersten Anhaltspunkt liefern.

*Unterstüt-
zung von
MC/DC*

Die Unterstützung des Überdeckungskriteriums MC/DC sollte ein weiteres Ziel weiterführender Arbeiten sein. Da dafür ein Satz von Testdaten erzeugt werden muss, müssen bei der Suche unterschiedliche Überdeckungsziele gleichzeitig betrachtet werden. Erste Arbeiten sind im Bereich des suchbasierten Tests für prozeduralen Code zu finden: Ghani und Clark verwenden zu diesem Zweck eine spezifische Zielfunktion [48]. Alternativ könnten die Algorithmen aus dem Bereich der multikriteriellen Optimierung verwendet werden [80].

Gegenstand weiterführender Arbeiten sollte zur Ermöglichung der weitreichenden praktischen Anwendbarkeit des entwickelten Verfahrens darüber hinaus die vollständige Unterstützung der SL/SF Semantik sein. Dazu zählen beispielsweise die bislang nicht unterstützten SF Funktionen.

KONZEPTUELLE ERWEITERUNGEN

Folgende Beispiele konzeptueller Erweiterungen des Verfahrens sind denkbar:

- Analyse der Variablenabhängigkeiten
- Modelltransformationen

Die Analyse der Variablenabhängigkeiten des Testobjekts (auch als *Program Slicing* bezeichnet) wird bei prozeduralem Code erfolgreich eingesetzt [59]. Der Testdatengenerierung vorgelagert, wird dabei statisch bestimmt, welche Überdeckungsziele von welchen Modelleingaben abhängen. Zusätzlich kann bestimmt werden, welche Wertebereiche der einzelnen Eingaben jeweils relevant sind. Dadurch kann der Suchraum für jedes Überdeckungsziel teilweise stark eingeschränkt und die Suche entsprechend verbessert werden. Es sollte in weiterführenden Arbeiten untersucht werden, inwieweit diese Methodik auf SL/SF Modelle anwendbar ist und zur Steigerung der Effizienz des suchbasierten Strukturtests verwendet werden kann.

*Model
Slicing*

Eine auf Slicing basierende Möglichkeit der Effizienzsteigerung könnte in der Transformation des Testobjekts bestehen. Mittels Slicing müsste zunächst analysiert werden, welche Modellteile für das jeweilige Überdeckungsziel relevant sind. Anschließend könnten alle nicht-relevanten Modellteile entfernt werden; ggf. würden Stubs notwendig. Es ergäbe sich ein deutlich kleineres Modell mit deutlich kürzerer Ausführungszeit. Ebenso sind Testability Transformations möglicherweise anwendbar. Bei einer *Testability Transformation* handelt es sich um eine spezifische Transformation des Testobjekts zur Erhöhung der Fähigkeit eines gegebenen Testdatengenerierungsverfahrens, Testdaten für das ursprüngliche Testobjekt zu generieren. Gegenstand der Transformation sind bekannte kritische strukturelle Konstrukte innerhalb des Testobjekts. Die Leistungsfähigkeit solcher Transformationen wurde für prozedurale Code vielfach nachgewiesen [60]. Weiterführende Arbeiten sollten sich demnach den Möglichkeiten umfassender Modelltransformationen zur Effizienz- und Effektivitätssteigerung des suchbasierten Strukturtests widmen.

*Modell-
transfor-
mationen*

WEITERE FORSCHUNGSRICHTUNGEN

Die Ergebnisse dieser Arbeit besitzen folgende Berührungspunkte mit anderen Bereichen der Testautomatisierung:

- Signalgenerierung für funktionsorientierte Tests
- Unterstützung mutationsbasierter Tests

*Funktionale
Tests*

Funktionsorientiertes Testen verifiziert das funktionale Verhalten des Testobjekts durch einen Widerspruchsbeweis. Es werden Testdaten gesucht, welche eine funktionale Anforderung des Testobjekts verletzen. Für die Automatisierung dieses Black-Box-Verfahrens ist die Plausibilität der generierten Testdaten von besonderer Relevanz. Der in dieser Arbeit entwickelte Signalgenerator konnte bereits erfolgreich für den funktionsorientierten Test eines realen Steuergeräts verwendet werden [86]. Weiterführende Arbeiten könnten sich jedoch mit einer Erweiterung des Signalgenerators für dieses Anwendungsfeld befassen. Beispielsweise könnte die enthaltene Constraintbehandlung zur Steuerung der Suche verwendet werden. Da das Brechen einer funktionalen Anforderung üblicherweise als Constraint formuliert werden kann, würde so die umständliche manuelle Bestimmung einer Zielfunktion entfallen.

*Mutations-
basierte
Tests*

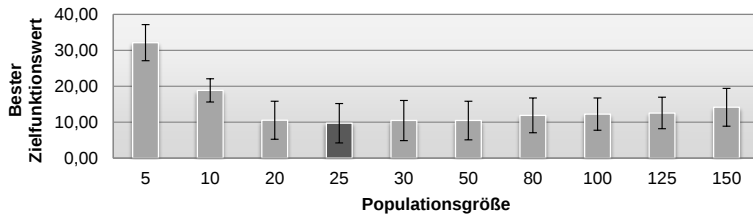
Mutationsbasierte Tests (engl. Mutation Testing) werden häufig zur Bewertung eines gegebenen Testdatensatzes durchgeführt. Mutationsbasierte Tests können jedoch auch dazu verwendet werden, einen Satz von Testdaten zu generieren, die ein hohes Potential zur Fehlerauffindung besitzen. Zu diesem Zweck wird das Testobjekt geringfügig verändert (mutiert). Derartigen Veränderungen sollen typische Programmierfehler imitieren: Es handelt sich beispielsweise um das Vertauschen von Variablennamen oder das einfache Wechseln eines Vorzeichens oder eines Operators. Das Verfahren sucht anschließend nach einem Testdatum, welches für das ursprüngliche und das mutierte Testobjekt ein unterschiedliches Verhalten hervorruft. Mutationsbasierte Tests sind auch bei SL/SF Modellen möglich [155]. Es sollte untersucht werden inwiefern die Ergebnisse dieser Arbeiten dazu genutzt werden können, den mutationsbasierten Test im industriellen Umfeld anwendbar zu machen.

Teil IV

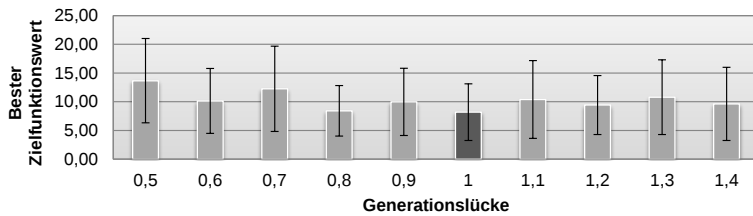
ANHANG

A.1 PARAMETER TUNING

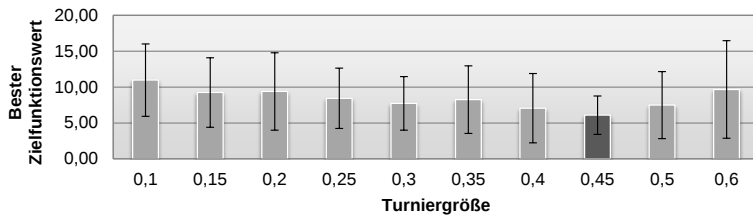
Dieser Abschnitt beschreibt die Ergebnisse einer experimentellen Untersuchung zur Bestimmung optimaler Parameterbelegungen für das Suchverfahren *LvSb* innerhalb von *SbGen-SL*.



(a) Optimale Belegung für den Parameter *Populationsgröße*: 25

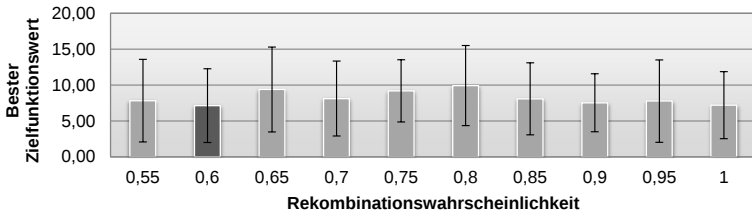


(b) Optimale Belegung für den Parameter *Generationslücke*: 1.0

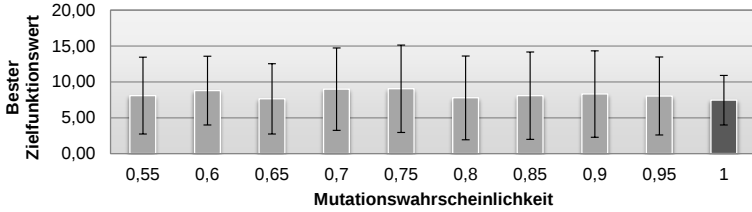


(c) Optimale Belegung für den Parameter *Turniergröße*: 0.45

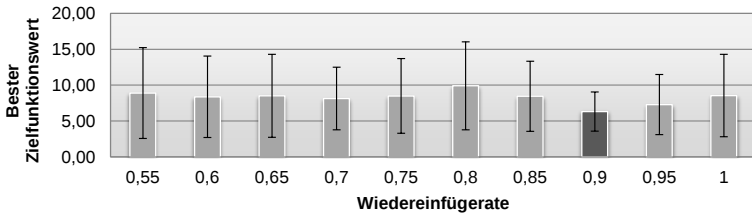
Abb. A.1: Ergebnisse der Experimente zur Bestimmung optimaler Belegungen für die Parameter von *SbGen-SL*.



(d) Optimale Belegung für den Parameter *Rekombinationswahrscheinlichkeit*: 0.6



(e) Optimale Belegung für den Parameter *Mutationswahrscheinlichkeit*: 1.0



(f) Optimale Belegung für den Parameter *Wiedereinfügerate*: 0.9

Abb. A.1 (Fs.): Ergebnisse der Experimente zur Bestimmung optimaler Belegungen für die Parameter von SBGen-SL.

A.2 ÜBERDECKUNGSZIELE

Die folgenden Abschnitte stellen die einzelnen strukturellen Überdeckungsziele für die untersuchten Testobjekte vor. Diese ergeben sich aus dem gewählten strukturellen Überdeckungskriterium der *Bedingungsüberdeckung*. Als Grundlage dienen die in Abschnitt 5.2.2 auf Seite 154 vorgestellten Testobjekte mitsamt der für Reactis Tester notwendigen Anpassungen.

Die aufgeführten Tabellen sind wie folgt strukturiert: Spalte *ID* führt eine kurze Bezeichnung des Überdeckungsziels auf. Die Spalte *Modellentität* bezeichnet das zugrundeliegende strukturelle Element des Modells – es handelt sich z.B. um einen SL Block

oder einen SF Zustand – für das eine bestimmte Bedingung erfüllt sein muss. Die Spalte *Bedingung* gibt diese notwendige Bedingung an und die letzte Spalte *Erreichbar* beschreibt die Erreichbarkeit des jeweiligen Überdeckungsziels. Grau hinterlegt ist jeweils der Pfad innerhalb des Modells zu den darunter aufgeführten Modellentitäten.

TESTOBJEKT T01

ID	Modellentität	Bedingung	Erreichbar
/Temperature Control Chart/			
tg_011	Blower_off	E == 1	•
tg_012	Blower_off	E == 0	
tg_013	Blower_off	in_temp_range(0,5,203)	•
tg_014	Blower_off	!in_temp_range(0,5,203)	•
tg_021	Blower_on	E == 1	•
tg_022	Blower_on	E == 0	
tg_023	Blower_on	absTempDiff>203	•
tg_024	Blower_on	absTempDiff<=203	•
tg_025	Blower_on	absTempDiff<=0.5	•
tg_026	Blower_on	absTempDiff>0.5	•
tg_0311	Blower_on	E == 1	•
tg_0312	Blower_on	E == 0	
tg_0313	Blower_on	!in(Blower_1)	•
tg_0314	Blower_on	in(Blower_1)	•
tg_0315	Blower_on	in_temp_range(1,5,5)	•
tg_0316	Blower_on	!in_temp_range(1,5,5)	•
tg_0321	Blower_on	!in(Blower_2)	•
tg_0322	Blower_on	in(Blower_2)	•
tg_0323	Blower_on	in_temp_range(7,10)	•
tg_0324	Blower_on	!in_temp_range(7,10)	•
tg_0331	Blower_on	!in(Blower_3)	•
tg_0332	Blower_on	in(Blower_3)	•
tg_0333	Blower_on	in_temp_range(12,15)	•
tg_0334	Blower_on	!in_temp_range(12,15)	•
tg_0341	Blower_on	!in(Blower_4)	•

tg_0342	Blower_on	in(Blower_4)	•
tg_0343	Blower_on	in_temp_range(17,20)	•
tg_0344	Blower_on	!in_temp_range(17,20)	•
tg_0351	Blower_on	!in(Blower_5)	•
tg_0352	Blower_on	in(Blower_5)	•
tg_0353	Blower_on	in_temp_range(22,200)	•
tg_0354	Blower_on	!in_temp_range(22,200)	•
tg_041	Heater_Act	E == 1	•
tg_042	Heater_Act	E == 0	
tg_043	Heater_Act	SetPTemp-IntTemp<=0.5	•
tg_044	Heater_Act	SetPTemp-IntTemp>0.5	•
tg_051	H_offAC_off	E == 1	•
tg_052	H_offAC_off	E == 0	
tg_053	H_offAC_off	SetPTemp-IntTemp>0.5	•
tg_054	H_offAC_off	SetPTemp-IntTemp<=0.5	•
tg_055	H_offAC_off	SetPTemp-IntTemp<-0.5	•
tg_056	H_offAC_off	SetPTemp-IntTemp>=-0.5	•
tg_061	ACAct	E == 1	•
tg_062	ACAct	E == 0	
tg_063	ACAct	SetPTemp-IntTemp>=-0.5	•
tg_064	ACAct	SetPTemp-IntTemp<-0.5	•
tg_071	Face	DistReq == 1	
tg_072	Face	DistReq == 0	•
tg_073	Feet	DistReq == 1	
tg_074	Feet	DistReq == 0	
tg_075	Defrost	DistReq == 1	
tg_076	Defrost	DistReq == 0	
tg_081	Recyc_on	RecycReq == 1	
tg_082	Recyc_on	RecycReq == 0	•
tg_083	Recyc_on	in(AirDist.Defrost)	
tg_084	Recyc_on	!in(AirDist.Defrost)	•
tg_085	Recyc_off	RecycReq == 1	
tg_086	Recyc_off	RecycReq == 0	

Tab. A.1: Strukturelle Überdeckungsziele von Testobjekt *TO1* für die Bedingungsüberdeckung.

TESTOBJEKT T02

ID	Modellentität	Bedingung	Erreichbar
/fuel rate controller/Airflow calculation/			
tg_0101	Logic1	In1 == 1	•
tg_0102	Logic1	In1 == 0	•
tg_0103	Logic1	In2 == 1	•
tg_0104	Logic1	In2 == 0	•
/fuel rate controller/Fuel Calculation/Switchable Compensation/			
tg_0201	Logical Operator	In1 == 1	•
tg_0202	Logical Operator	In1 == 0	•
tg_0203	Logical Operator	In2 == 1	•
tg_0204	Logical Operator	In2 == 0	•
/fuel rate controller/control logic/			
tg_0301	O2_fail	Ego < max_ego	
tg_0302	O2_fail	Ego >= max_ego	
tg_0303	O2_warmup	t > o2_t_thresh	•
tg_0304	O2_warmup	t <= o2_t_thresh	•
tg_0305	O2_normal	Ego > max_Ego	
tg_0306	O2_normal	Ego <= max_Ego	•
tg_0307	press_norm	press > max_press	•
tg_0308	press_norm	press <= max_press	•
tg_0309	press_norm	press < min_press	
tg_0310	press_norm	press >= min_press	•
tg_0311	press_fail	press > min_press	•
tg_0312	press_fail	press <= min_press	•
tg_0313	press_fail	press < max_press	•
tg_0314	press_fail	press >= max_press	•
tg_0315	throt_norm	throt > max_throt	
tg_0316	throt_norm	throt <= max_throt	•
tg_0317	throt_norm	throt < min_throt	
tg_0318	throt_norm	throt >= min_throt	•
tg_0319	throt_fail	throt > min_throt	
tg_0320	throt_fail	throt <= min_throt	
tg_0321	throt_fail	throt < max_throt	

tg_0322	throt_fail	throt >= max_throt	
tg_0323	speed_norm	speed == 0	
tg_0324	speed_norm	speed != 0	•
tg_0325	speed_norm	press < zero_thresh	•
tg_0326	speed_norm	press >= zero_thresh	•
tg_0327	speed_fail	speed > 0	
tg_0328	speed_fail	speed <= 0	
tg_0329	FL0	INC	•
tg_0330	FL0	!INC	•
tg_0331	FL1	INC	
tg_0332	FL1	!INC	•
tg_0333	FL1	DEC	•
tg_0334	FL1	!DEC	•
tg_0335	FL2	INC	
tg_0336	FL2	!INC	
tg_0337	FL2	DEC	
tg_0338	FL2	!DEC	
tg_0339	FL3	INC	
tg_0340	FL3	!INC	
tg_0341	FL3	DEC	
tg_0342	FL3	!DEC	
tg_0343	FL4	DEC	
tg_0344	FL4	!DEC	
tg_0345	conn	!in(MultiFail)	
tg_0346	conn	in(MultiFail)	
tg_0347	overspeed	in(speed_norm)	
tg_0348	overspeed	!in(speed_norm)	
tg_0349	overspeed	speed < max_speed - hys	
tg_0350	overspeed	speed >= max_speed - hys	
tg_0351	shutdown	in(FL1)	
tg_0352	shutdown	!in(FL1)	
tg_0353	Running	speed > max_speed	
tg_0354	Running	speed <= max_speed	•
tg_0355	Running	in(FL2)	
tg_0356	Running	!in(FL2)	•
tg_0357	conn	in(FL1)	•

tg_0358	conn	!in(FL1)	•
tg_0359	Normal	in(FL1)	•
tg_0360	Normal	!in(FL1)	•
tg_0361	Warmup	in(O2_normal)	•
tg_0362	Warmup	!in(O2_normal)	•
tg_0363	Single_Failure	in(FL0)	•
tg_0364	Single_Failure	!in(FL0)	•

Tab. A.2: Strukturelle Überdeckungsziele von Testobjekt *TO2* für die Bedingungsüberdeckung.

TESTOBJEKT TO3

ID	Modellentität	Bedingung	Erreichbar
/shift_logic/			
tg_011	steady_state	speed > up_th	•
tg_012	steady_state	speed <= up_th	•
tg_013	steady_state	speed < down_th	•
tg_014	steady_state	speed >= down_th	•
tg_021	upshifting	speed >= up_th	•
tg_022	upshifting	speed < up_th	•
tg_023	upshifting	tick >= 2	•
tg_024	upshifting	tick < 2	•
tg_031	downshifting	speed <= down_th	•
tg_032	downshifting	speed > down_th	•
tg_033	downshifting	tick >= 2	•
tg_034	downshifting	tick < 2	•

Tab. A.3: Strukturelle Überdeckungsziele von Testobjekt *TO3* für die Bedingungsüberdeckung.

TESTOBJEKT TO4

Die einzelnen Überdeckungsziele für Testobjekt *TO4* sind hier aus Gründen der Geheimhaltung nicht aufgeführt.

A.3 FALLSTUDIENERGEBNISSE

Die folgenden Abschnitte beschreiben die Ergebnisse der Fallstudie. Vergleichend dargestellt ist für die untersuchten Testdatengenerierungsverfahren die durchschnittliche Erreichung der einzelnen strukturellen Überdeckungsziele der vier Testobjekte. Diese durchschnittliche Erreichung der Überdeckungsziele wird dabei als direktes Maß für die Effektivität der Verfahren betrachtet.

TESTOBJEKT TO1

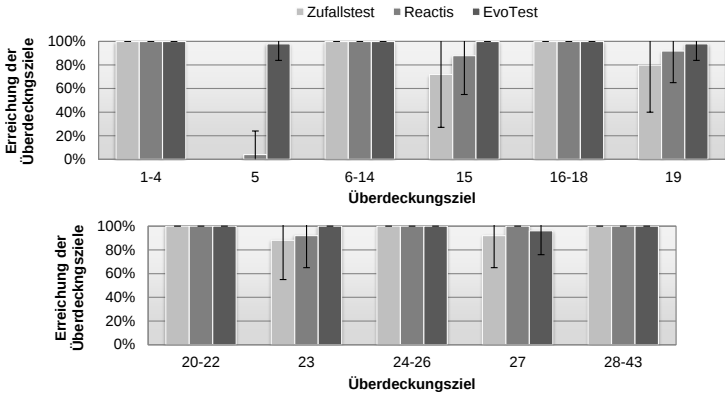


Abb. A.2: Vergleich der Testdatengeneratoren bzgl. der durchschnittlichen Erreichung aller Überdeckungsziele von Testobjekt TO1 (gemittelt über jeweils 50 Experimente).

TESTOBJEKT TO2

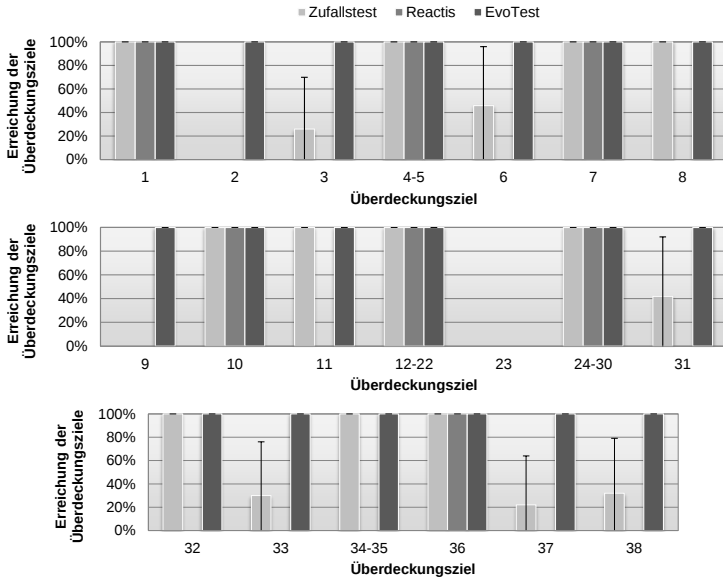


Abb. A.3: Vergleich der Testdatengeneratoren bzgl. der durchschnittlichen Erreichung aller Überdeckungsziele von Testobjekt TO₂ (gemittelt über jeweils 50 Experimente).

TESTOBJEKT TO3

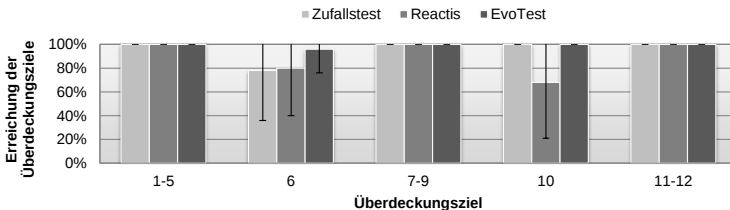


Abb. A.4: Vergleich der Testdatengeneratoren bzgl. der durchschnittlichen Erreichung aller Überdeckungsziele von Testobjekt TO₃ (gemittelt über jeweils 50 Experimente).

TESTOBJEKT TO4

Die Ergebnisse für die einzelnen Überdeckungsziele für Testobjekt TO₄ sind hier aus Gründen der Geheimhaltung nicht aufgeführt.

LITERATURVERZEICHNIS

- [1] A. Agrawal, G. Simon und G. Karsai (2004). *Semantic Translation of Simulink/Stateflow Models to Hybrid Automata Using Graph Transformations*. Electronic Notes in Theoretical Computer Science, **109**, 43–56. ISSN 1571-0661. (Zitiert auf Seite 28.)
- [2] R. Alur und T. A. Henzinger (1992). *Logics and Models of Real Time: A Survey*. In Proceedings of the Real-Time: Theory in Practice, REX Workshop, Seiten 74–106. Springer-Verlag, London (GB). ISBN 3-540-55564-1. (Zitiert auf Seite 93.)
- [3] R. Alur, T. Feder und T. A. Henzinger (1996). *The Benefits of Relaxing Punctuality*. Journal of the ACM, **43**(1), 116–146. ISSN 0004-5411. (Zitiert auf Seite 94.)
- [4] R. Alur, A. Kanade, S. Ramesh und K. C. Shashidhar (2008). *Symbolic Analysis for Improving Simulation Coverage of Simulink/Stateflow Models*. In Proceedings of the 8th ACM International Conference on Embedded Software, Seiten 89–98. ACM, New York, NY, USA. ISBN 978-1-60558-468-3. (Zitiert auf den Seiten 28, 43 und 44.)
- [5] A. Arcuri (2010). *It Does Matter How You Normalise the Branch Distance in Search Based Software Testing*. In Proceedings of the 3rd International Conference on Software Testing, Verification and Validation (ICST), Seiten 205–214. (Zitiert auf Seite 36.)
- [6] E. Asarin, P. Caspi und O. Maler (2002). *Timed Regular Expressions*. Journal of the ACM, **49**(2), 172–206. ISSN 0004-5411. (Zitiert auf Seite 93.)
- [7] J. E. Baker (1987). *Reducing Bias and Inefficiency in the Selection Algorithm*. In Proceedings of the 2nd International Conference on Genetic Algorithms and their Application, Seiten 14–21. Lawrence Erlbaum Associates, Inc., Mahwah, NJ, USA. ISBN 0-8058-0158-8. (Zitiert auf Seite 54.)

- [8] C. Banphawatthanarak, B. Krogh und K. Butts (1999). *Symbolic Verification of Executable Control Specifications*. In Proceedings of the IEEE International Symposium on Computer Aided Control System Design, Seiten 581–586. ISBN 0-7803-5500-8. (Zitiert auf Seite 28.)
- [9] W. Banzhaf (1993). *Genetic Programming for Pedestrians*. In Proceedings of the 5th International Conference on Genetic Algorithms, Seite 628. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. ISBN 1-55860-299-2. (Zitiert auf Seite 58.)
- [10] W. Banzhaf, P. Nordin, R. E. Keller und F. D. Francone (1998). *Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications*. Morgan Kaufmann, San Francisco, CA, USA. ISBN 3-920993-58-6. (Zitiert auf Seite 58.)
- [11] A. Baresel (2000). *Automatisierung von Strukturtests mit evolutionären Algorithmen*. Diplomarbeit, Humboldt Universität, Berlin. (Zitiert auf Seite 36.)
- [12] A. Baresel, M. Conrad, S. Sadeghipour und J. Wegener (2003). *The Interplay between Model and Code Coverage*. In Proceedings of the 11th European International Conference on Software Testing, Analysis and Review. Amsterdam, Netherlands. (Zitiert auf Seite 19.)
- [13] A. Baresel, H. Pohlheim und S. Sadeghipour (2003). *Structural and Functional Sequence Test of Dynamic and State-Based Software with Evolutionary Algorithms*. In Proceedings of the 5th Genetic and Evolutionary Computation Conference, Seiten 2428–2441. (Zitiert auf Seite 72.)
- [14] P. Bechberger (2000). *Modellbasierte Softwareentwicklung für Steuergeräte*. Automobiltechnische Zeitschrift, Sonderausgabe Automotive Electronics, Seiten 16–24. (Zitiert auf Seite 3.)
- [15] B. Beizer (1990). *Software testing techniques (2nd ed.)*. Van Nostrand Reinhold Co., New York, NY, USA. ISBN 0-442-20672-0. (Zitiert auf den Seiten 4 und 21.)
- [16] H.-G. Beyer (2001). *The theory of Evolution Strategies*. Springer, Berlin. ISBN 978-3-540-67297-5. (Zitiert auf Seite 56.)

- [17] C. Bigot, A. Faivre, J.-P. Gallois, A. Lapitre, D. Lugato, J.-Y. Pierron und N. Rapin (2003). *Automatic Test Generation with AGATHA*. In Proceedings of the 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Seiten 591–596. Springer-Verlag, Berlin, Heidelberg. ISBN 3-540-00898-5. (Zitiert auf den Seiten 28 und 29.)
- [18] B. W. Boehm (1988). *A Spiral Model of Software Development and Enhancement*. Computer, **21**(5), 61–72. ISSN 0018-9162. (Zitiert auf Seite 17.)
- [19] L. Bottaci (2003). *Predicate Expression Cost Functions to Guide Evolutionary Search for Test Data*. In Proceedings of the 5th Conference on Genetic and Evolutionary Computation, Seiten 2455–2464. Chicago, Illinois, USA. (Zitiert auf den Seiten 36 und 39.)
- [20] C. Bourhfir, R. Dssouli, E. M. Aboulhamid und N. Rico (1997). *Automatic Executable Test Case Generation for Extended Finite State Machine Protocols*. In Proceedings of the International Workshop on Testing Communicating Systems, Seiten 75–90. (Zitiert auf Seite 28.)
- [21] R. S. Boyer, B. Elspas und K. N. Levitt (1975). *SELECT – A Formal System for Testing and Debugging Programs by Symbolic Execution*. In Proceedings of the International Conference on Reliable Software, Seiten 234–245. ACM, New York, NY, USA. (Zitiert auf Seite 24.)
- [22] M. Brameier (2004). *On Linear Genetic Programming*. Dissertation, Fachbereich Informatik, Universität Dortmund, Germany. URL <https://eldorado.uni-dortmund.de/bitstream/2003/20098/1/Brameier.ps>. (Zitiert auf den Seiten 58, 59 und 61.)
- [23] E. Bringmann und A. Krämer (2008). *Model-Based Testing of Automotive Systems*. In Proceedings of the 1st International Conference on Software Testing, Verification, and Validation, Seiten 485–493. (Zitiert auf Seite 3.)
- [24] F. E. Cellier (1991). *Continuous System Modeling*. Springer-Verlag New York, Inc., Secaucus, NJ, USA. ISBN 0387975020. (Zitiert auf Seite 12.)

- [25] S. T. Chanson und J. Zhu (1994). *Automatic Protocol Test Suite Derivation*. In Proceedings of the 13th IEEE Conference on Networking for Global Communications, Band 2, Seiten 792–799. (Zitiert auf Seite 28.)
- [26] J. Chilenski und S. Miller (1994). *Applicability of Modified Condition/Decision Coverage to Software Testing*. Software Engineering Journal, 9(5), 193–200. ISSN 0268-6961. (Zitiert auf Seite 20.)
- [27] L. A. Clarke (1976). *A System to Generate Test Data and Symbolically Execute Programs*. IEEE Transactions on Software Engineering, 2(3), 215–222. ISSN 0098-5589. (Zitiert auf den Seiten 5 und 24.)
- [28] C. A. Coello (2002). *Theoretical and Numerical Constraint-Handling Techniques used with Evolutionary Algorithms: A Survey of the State of the Art*. Computer Methods in Applied Mechanics and Engineering, 191(11–12), 1245–1287. ISSN 0045-7825. (Zitiert auf den Seiten 99 und 101.)
- [29] M. Conrad (2004). *Auswahl und Beschreibung von Testszenarien für den Modell-basierten Test eingebetteter Software im Automobil*. Dissertation, Technische Universität Berlin. (Zitiert auf Seite 15.)
- [30] M. Conrad und H. Dörr (2006). *Model-Based Development of In-Vehicle Software*. In Proceedings of the Conference on Design, Automation and Test in Europe, Seiten 89–90. European Design and Automation Association, 3001 Leuven, Belgium, Belgium. ISBN 3-9810801-0-6. (Zitiert auf Seite 4.)
- [31] M. Conrad und I. Fey (2005). *Modell-basierter Test von Simulink/Stateflow-Modellen*. In Kolloquium Testen im System- und Software-Life-Cycle, Technische Akademie Esslingen, Seiten 278–298. (Zitiert auf den Seiten 4, 11 und 18.)
- [32] R. Courant (1943). *Variational Methods for the Solution of Problems of Equilibrium and Vibrations*. Bulletin of the American Mathematical Society. (Zitiert auf Seite 99.)
- [33] N. L. Cramer (1985). *A Representation for the Adaptive Generation of Simple Sequential Programs*. In Proceedings of the 1st International Conference on Genetic Algorithms, Seiten 183–187. L. Erlbaum Associates Inc., Hillsdale, NJ, USA. ISBN 0-8058-0426-9. (Zitiert auf den Seiten 57 und 58.)

- [34] J. B. Dabney und T. L. Harman (1997). *Mastering Simulink*. Prentice Hall PTR, Upper Saddle River, NJ, USA. ISBN 0132437678. (Zitiert auf Seite 14.)
- [35] W. H. Deason, D. B. Brown, K.-H. Chang und J. H. Cross (1991). *A Rule-Based Software Test Data Generator*. IEEE Transactions on Knowledge and Data Engineering, 3(1), 108–117. ISSN 1041-4347. (Zitiert auf den Seiten 4 und 17.)
- [36] E. W. Dijkstra (1972). *Chapter I: Notes on Structured Programming*. Structured Programming, Seiten 1–82. (Zitiert auf Seite 17.)
- [37] W. Dröschel und M. Wiemers (1999). *Das V-Modell 97: Der Standard für die Entwicklung von IT-Systemen mit Anleitung für den Praxiseinsatz*. Oldenbourg. ISBN 978-3486250862. (Zitiert auf Seite 17.)
- [38] A. E. Eiben und J. E. Smith (2003). *Introduction to Evolutionary Computing (Natural Computing Series)*. Springer. ISBN 3-540-40184-9. (Zitiert auf Seite 56.)
- [39] ETAS GmbH (2010). *ASCET® (Advanced Simulation and Control Engineering Tool)*. <http://www.etas.com/ascet>. Letzter Zugriff: März 2011. (Zitiert auf Seite 4.)
- [40] Florin Pinte, Francesca Saglietti und Norbert Oster (2008). *Automatic Generation of Optimized Integration Test Data by Genetic Algorithms*. In Walid Maalej und Bernd Brügge, Herausgeber, Software Engineering 2008 - Workshopband, Seiten 415–422. Lecture Notes in Informatics (LNI), Gesellschaft für Informatik (GI) e. V., Bonn. ISBN 978-3-88579-216-1. (Zitiert auf Seite 40.)
- [41] D. B. Fogel (1994). *An Introduction to Simulated Evolutionary Optimization*. IEEE Transactions on Neural Networks, 5(1), 3–14. (Zitiert auf Seite 48.)
- [42] L. J. Fogel, A. J. Owens und M. J. Walsh (1966). *Artificial Intelligence through Simulated Evolution*. John Wiley, New York, USA. (Zitiert auf Seite 50.)
- [43] K. Fuest und A. Pols (2007). *Zukunft digitale Wirtschaft*. Gemeinsame Studie des BITKOM e.V. und der Roland Berger Strategy Consultants GmbH. (Zitiert auf Seite 3.)

- [44] A. A. Gadkari, S. Mohalik, K. Shashidhar, A. Yeolekar, J. Suresh und S. Ramesh (2007). *Automatic Generation of Test-Cases Using Model Checking for SL/SF Models*. In Proceedings of the 4th Model-Driven Engineering, Verification and Validation Workshop, Seiten 33–46. (Zitiert auf Seite 27.)
- [45] A. A. Gadkari, A. Yeolekar, J. Suresh, S. Ramesh, S. Mohalik und K. C. Shashidhar (2008). *AutoMOTGen: Automatic Model Oriented Test Generator for Embedded Control Systems*. In Proceedings of the 20th International Conference on Computer Aided Verification, Seiten 204–208. Springer-Verlag, Berlin, Heidelberg. ISBN 978-3-540-70543-7. (Zitiert auf Seite 27.)
- [46] M. R. Garey und D. S. Johnson (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman. ISBN 0-7167-1044-7. (Zitiert auf Seite 48.)
- [47] A. Gargantini und C. Heitmeyer (1999). *Using Model Checking to Generate Tests from Requirements Specifications*. ACM SIGSOFT Software Engineering Notes, **24**(6), 146–162. ISSN 0163-5948. (Zitiert auf Seite 27.)
- [48] K. Ghani und J. A. Clark (2009). *Automatic Test Data Generation for Multiple Condition and MCDC Coverage*. In Proceedings of the 4th International Conference on Software Engineering Advances, Seiten 152 –157. (Zitiert auf Seite 184.)
- [49] K. Ghani, J. A. Clark und Y. Zhan (2009). *Comparing Algorithms for Search-Based Test Data Generation of Matlab®Simulink®Models*. In Proceedings of the 11th Conference on Evolutionary Computation, Seiten 2940–2947. IEEE Press, Piscataway, NJ, USA. ISBN 978-1-4244-2958-5. (Zitiert auf Seite 39.)
- [50] P. Godefroid, N. Klarlund und K. Sen (2005). *DART: directed automated random testing*. ACM SIGPLAN Notices, **40**(6), 213–223. ISSN 0362-1340. (Zitiert auf Seite 30.)
- [51] D. E. Goldberg und K. Deb (1990). *A Comparative Analysis of Selection Schemes Used in Genetic Algorithms*. In Foundations of Genetic Algorithms, Seiten 69–93. (Zitiert auf Seite 55.)

- [52] D. R. Graham (1991). *Software Testing Tools: A New Classification Scheme*. Software Testing, Verification and Reliability, **1**(3), 17–34. (Zitiert auf Seite 4.)
- [53] A. Griep (2002). *Kompakte Beschreibung von Testsequenzen für die automatisierte Testfallerstellung unter Beachtung regelungstechnischer Aspekte*. Diplomarbeit, Technische Universität Ilmenau, Ilmenau. (Zitiert auf den Seiten 72 und 74.)
- [54] K. Grimm (1995). *Systematisches Testen von Software: Eine neue Methode und eine effektive Teststrategie*. Dissertation, Technische Universität Berlin. (Zitiert auf Seite 18.)
- [55] M. Grochtmann (2000). *Einführung in den systematischen Test*. Präsentation auf der Euroforum Konferenz. (Zitiert auf Seite 4.)
- [56] R. Grosu und S. A. Smolka (2005). Monte carlo model checking. In N. Halbwachs und L. D. Zuck, Herausgeber, Tools and Algorithms for the Construction and Analysis of Systems, Band 3440 von *Lecture Notes in Computer Science*, Seiten 271–286. Springer Berlin / Heidelberg. URL http://dx.doi.org/10.1007/978-3-540-31980-1_18. 10.1007/978-3-540-31980-1_18. (Zitiert auf Seite 42.)
- [57] D. Harel (1987). *Statecharts: A visual formalism for complex systems*. Science of Computer Programming, **8**(3), 231–274. ISSN 0167-6423. (Zitiert auf Seite 16.)
- [58] M. Harman und P. McMinn (2010). *A Theoretical and Empirical Study of Search-Based Testing: Local, Global, and Hybrid Search*. IEEE Transactions on Software Engineering, **36**(2), 226–247. ISSN 0098-5589. (Zitiert auf Seite 169.)
- [59] M. Harman, C. Fox, R. M. Hierons, L. Hu, S. Danicic und J. Wegener (2002). *VADA: A Transformation-based System for Variable Dependence Analysis*. IEEE International Workshop on Source Code Analysis and Manipulation, Seiten 55–64. (Zitiert auf Seite 185.)
- [60] M. Harman, L. Hu, R. Hierons, J. Wegener, H. Sthamer, A. Baresel und M. Roper (2004). *Testability Transformation*. IEEE Transactions on Software Engineering, **30**(1), 3–16. ISSN 0098-5589. (Zitiert auf Seite 185.)

- [61] M. S. Hecht (1977). *Flow Analysis of Computer Programs*. Elsevier Science Inc., New York, NY, USA. ISBN 0444002162. (Zitiert auf Seite 20.)
- [62] A. Helmerich, N. Koch, L. Mandel, P. Braun, P. Dornbusch, A. Gruler, P. Keil, R. Leisibach, J. Romberg, B. Schätz, T. Wild und G. Wimmel (2005). *Study of Worldwide Trends and R&D Programmes in Embedded Systems in View of Maximising the Impact of a Technology Platform in the Area*. Final Report for the European Commission, Brüssel, Belgien. (Zitiert auf Seite 4.)
- [63] J. H. Holland (1962). *Outline for a Logical Theory of Adaptive Systems*. Journal of the Association for Computing Machinery, 9(3), 297–314. (Zitiert auf Seite 49.)
- [64] J. H. Holland (1975). *Adaption in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, Michigan. (Zitiert auf Seite 49.)
- [65] H. S. Hong, I. Lee und O. Sokolsky (2001). *Automatic Test Generation from Statecharts Using Model Checking*. In Proceedings of the Workshop on Formal Approaches to Testing of Software, Volume NS-01-4 of BRICS Notes Series, Seiten 15–30. (Zitiert auf Seite 28.)
- [66] H. S. Hong, I. Lee, O. Sokolsky und H. Ural (2002). *A Temporal Logic Based Theory of Test Coverage and Generation*. In Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Seiten 327–341. Springer-Verlag, London, UK. ISBN 3-540-43419-4. (Zitiert auf Seite 28.)
- [67] W. E. Howden (1977). *Symbolic Testing and the DISSECT Symbolic Evaluation System*. IEEE Transactions on Software Engineering, 3(4), 266–278. ISSN 0098-5589. (Zitiert auf den Seiten 5 und 24.)
- [68] J. C. Huang (1975). *An Approach to Program Testing*. ACM Computing Surveys, 7(3), 113–128. ISSN 0360-0300. (Zitiert auf Seite 24.)
- [69] D. C. Ince (1987). *The Automatic Generation of Test Data*. The Computer Journal, 30(1), 63–69. URL <http://comjnl.oxfordjournals.org/cgi/content/abstract/30/1/63>. (Zitiert auf Seite 4.)

- [70] International Organization for Standardization (ISO) (1999). ISO C Standard 1999. Technischer Bericht, International Organization for Standardization. URL <http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1124.pdf>. ISO/IEC 9899:1999 draft. (Zitiert auf Seite 31.)
- [71] International Organization for Standardization (ISO) (2009). *ISO/DIS 26262-6: Road vehicles – Functional safety – Part 6: Product development: software level*. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=51362. Letzter Zugriff: Mai 2010. (Zitiert auf den Seiten 5 und 17.)
- [72] International Organization for Standardization (ISO) (2009). *ISO/DIS 26262-9: Road vehicles – Functional safety – Part 9: ASIL-oriented and safety-oriented analyses*. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=51365. Letzter Zugriff: Mai 2010. (Zitiert auf den Seiten 5 und 17.)
- [73] A. Kalaji, R. M. Hierons und S. Swift (2009). *Generating Feasible Transition Paths for Testing from an Extended Finite State Machine (EFSM)*. In *Proceedings of the International Conference on Software Testing Verification and Validation*, Seiten 230–239. IEEE Computer Society, Washington, DC, USA. ISBN 978-0-7695-3601-9. (Zitiert auf Seite 42.)
- [74] J. Kennedy und R. C. Eberhart (2001). *Swarm Intelligence*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. ISBN 1-55860-595-9. (Zitiert auf Seite 49.)
- [75] J. C. King (1976). *Symbolic Execution and Program Testing*. *Communications of the ACM*, **19**(7), 385–394. ISSN 0001-0782. (Zitiert auf den Seiten 5 und 24.)
- [76] S. Kirkpatrick, C. D. Gelatt und M. P. Vecchi (1983). *Optimization by Simulated Annealing*. *Science*, Number 4598, **220**, 4598, 671–680. (Zitiert auf Seite 39.)
- [77] B. Kitchenham, S. Pfleeger, L. Pickard, P. Jones, D. Hoaglin, K. El Emam und J. Rosenberg (2002). *Preliminary Guidelines for Empirical Research in Software Engineering*. *IEEE Transactions on Software Engineering*, **28**(8), 721 – 734. ISSN 0098-5589. (Zitiert auf Seite 149.)

- [78] B. Korel (1990). *Automated Software Test Data Generation*. IEEE Transactions on Software Engineering, **16**(8), 870–879. ISSN 0098-5589. (Zitiert auf Seite 31.)
- [79] J. R. Koza (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection (Complex Adaptive Systems)*. The MIT Press. ISBN 0262111705. (Zitiert auf den Seiten 49 und 57.)
- [80] K. Lakhotia, M. Harman und P. McMinn (2007). *A Multi-Objective Approach To Search-Based Test Data Generation*. In Proceedings of the 9th annual Conference on Genetic and Evolutionary Computation, Seiten 1098–1105. ACM, New York, NY, USA. ISBN 978-1-59593-697-4. URL <http://doi.acm.org/10.1145/1276958.1277175>. (Zitiert auf Seite 184.)
- [81] W. B. Langdon und W. Banzhaf (2004). *Repeated Sequences in Linear Genetic Programming Genomes*. In Late Breaking Papers at the 2004 Genetic and Evolutionary Computation Conference. AAAI, Seattle, Washington, USA. URL <http://www.cs.bham.ac.uk/~wbl/biblio/gecco2004/LBP048.pdf>. (Zitiert auf Seite 59.)
- [82] E. Larson und T. Austin (2003). *High Coverage Detection of Input-Related Security Facults*. In Proceedings of the 12th Conference on USENIX Security Symposium, Seiten 9–9. USENIX Association, Berkeley, CA, USA. (Zitiert auf Seite 29.)
- [83] R. Lefticaru und F. Ipate (2007). *Automatic State-Based Test Generation Using Genetic Algorithms*. In SYNASC '07: Proceedings of the Ninth International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, Seiten 188–195. IEEE Computer Society, Washington, DC, USA. ISBN 0-7695-3078-8. (Zitiert auf den Seiten 40, 41 und 42.)
- [84] R. Lefticaru und F. Ipate (2008). *Functional Search-Based Testing from State Machines*. In Proceedings of the 1st International Conference on Software Testing, Verification, and Validation, Seiten 525–528. (Zitiert auf Seite 41.)
- [85] P. Liggesmeyer (1990). *Modultest und Modulverifikation: State of the Art*. Angewandte Informatik; Band 4, BI-Wissenschaftsverlag, Mannheim. ISBN 3-411-14361-4. (Zitiert auf Seite 5.)

- [86] F. Lindlar und A. Windisch (2010). *A Search-Based Approach to Functional Hardware-in-the-Loop Testing*. In Proceedings of the 2nd International Symposium on Search Based Software Engineering, Seiten 111–119. (Zitiert auf Seite 186.)
- [87] D. Lugato, C. Bigot, Y. Valot, J.-P. Gallois, S. Gérard und F. Terrier (2004). *Validation and Automatic Test Generation on UML Models: the AGATHA Approach*. International Journal on Software Tools for Technology Transfer, 5(2), 124–139. ISSN 1433-2779. (Zitiert auf Seite 29.)
- [88] R. Majumdar und K. Sen (2007). *Hybrid Concolic Testing*. In Proceedings of the 29th International Conference on Software Engineering, Seiten 416–426. IEEE Computer Society, Washington, DC, USA. ISBN 0-7695-2828-7. (Zitiert auf Seite 30.)
- [89] O. Maler und D. Nickovic (2004). *Monitoring Temporal Properties of Continuous Signals*. In Formal Modeling and Analysis of Timed Systems. Grenoble, France. (Zitiert auf den Seiten 93, 97, 98 und 117.)
- [90] O. Maler und A. Pnueli (2005). *Extending PSL for Analog Circuits*. Research Report (Deliverable D1.3/1) of Project PROSYD (Property-Based System Design). (Zitiert auf den Seiten 94 und 99.)
- [91] O. Maler, D. Nickovic und A. Pnueli (2006). *From MITL to Timed Automata*. In Proceedings of the 4th International Conference on Formal Modeling and Analysis of Timed Systems, Band 4202 von *Lecture Notes in Computer Science*, Seiten 274–289. Springer-Verlag, Berlin, Heidelberg. ISBN 3-540-45026-2. (Zitiert auf Seite 93.)
- [92] O. Maler, D. Nickovic und A. Pnueli (2008). *Checking Temporal Properties of Discrete, Timed and Continuous Behaviors*. In Pillars of Computer Science, Seiten 475–505. Lecture Notes in Computer Science, Springer. ISBN 978-3-540-78126-4. (Zitiert auf Seite 94.)
- [93] P. McMinn (2004). *Search-Based Software Test Data Generation: A Survey: Research Articles*. Software Testing, Verification & Reliability, 14(2), 105–156. ISSN 0960-0833. (Zitiert auf den Seiten 21, 31 und 179.)

- [94] P. McMinn und M. Holcombe (2005). *Evolutionary Testing of State-Based Programs*. In Proceedings of the 7th Conference on Genetic and Evolutionary Computation, Seiten 1013–1020. ACM, New York, NY, USA. ISBN 1-59593-010-8. (Zitiert auf Seite 36.)
- [95] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller und E. Teller (1953). *Equation of State Calculations by Fast Computing Machines*. The Journal of Chemical Physics, **21**(6), 1087–1092. URL <http://dx.doi.org/10.1063/1.1699114>. (Zitiert auf Seite 39.)
- [96] Z. Michalewicz (1995). *A Survey of Constraint Handling Techniques in Evolutionary Computation Methods*. In Proceedings of the 4th Annual Conference on Evolutionary Programming, Seiten 135–155. MIT Press. (Zitiert auf Seite 99.)
- [97] Z. Michalewicz und M. Schoenauer (1996). *Evolutionary Algorithms for Constrained Parameter Optimization Problems*. Evolutionary Computation, **4**, 1–32. (Zitiert auf Seite 99.)
- [98] M. Minsky (1965). *Models, Minds, Machines*. In Proceedings of the International Federation for Information Processing World Computer Congress, Seiten 45–49. (Zitiert auf Seite 12.)
- [99] National Instruments Corporation (2010). *LabVIEW® (Laboratory Virtual Instrumentation Engineering Workbench)*. <http://www.ni.com/labview>. Letzter Zugriff: März 2010. (Zitiert auf Seite 4.)
- [100] D. Nickovic, O. Maler, A. Pnueli, P. Caspi und A. Girard (2007). *Final Proposal for PSL Analog Extensions*. Forschungsbericht D1.3/2 des Projektes PROSYD. (Zitiert auf den Seiten 94 und 99.)
- [101] Norbert Oster (2007). *Automatische Generierung optimaler struktureller Testdaten für objekt-orientierte Software mittels multi-objektiver Metaheuristiken*. Dissertation, Friedrich-Alexander-Universität Erlangen-Nürnberg, Erlangen. (Zitiert auf Seite 40.)
- [102] Norbert Oster, Claudia Schieber, Francesca Saglietti und Florin Pinte (2007). *Automatische, modellbasierte Testdatengenerierung durch Einsatz evolutionärer Verfahren*. In Rainer

- Koschke, Otthein Herzog, Karl-Heinz Rödiger, und Marc Ronthaler, Herausgeber, Informatik 2007 - Informatik trifft Logistik, Band 2 von *Lecture Notes in Informatics*, Seiten 398–403. Köllen Druck+Verlag GmbH, Bonn. ISBN 978-3-88579-204-8. (Zitiert auf den Seiten 40 und 48.)
- [103] P. Nordin (1994). *A Compiling Genetic Programming System that Directly Manipulates the Machine Code*. *Advances in Genetic Programming*, Seiten 311–331. (Zitiert auf Seite 58.)
- [104] P. Nordin, W. Banzhaf und F. D. Francone (1999). *Efficient Evolution of Machine Code for CISC Architectures using Blocks and Homologous Crossover*. In *Advances in Genetic Programming III*, Seiten 275–299. MIT Press, Cambridge, MA. (Zitiert auf Seite 59.)
- [105] T. Perks (1994). *Stack-Based Genetic Programming*. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, Band 1, Seiten 148–153. IEEE Press, Orlando, Florida, USA. (Zitiert auf Seite 58.)
- [106] D. E. Perry, A. A. Porter und L. G. Votta (2000). *Empirical Studies of Software Engineering: A Roadmap*. In *Proceedings of the Conference on The Future of Software Engineering*, Seiten 345–355. ACM, New York, NY, USA. ISBN 1-58113-253-0. URL <http://doi.acm.org/10.1145/336512.336586>. (Zitiert auf Seite 149.)
- [107] Prover Technology (2010). *Prover[®] Plug-In*. http://www.prover.com/products/prover_plugin. Letzter Zugriff: Oktober 2010. (Zitiert auf Seite 44.)
- [108] C. S. Păsăreanu und W. Visser (2009). *A Survey of New Trends in Symbolic Execution for Software Testing and Analysis*. *International Journal on Software Tools for Technology Transfer*, **11**(4), 339–353. ISSN 1433-2779. (Zitiert auf den Seiten 6, 31, 45 und 47.)
- [109] C. S. Păsăreanu, J. Schumann, P. Mehrlitz, M. Lowry, G. Karsai, H. Nine und S. Neema (2009). *Model Based Analysis and Test Generation for Flight Software*. In *Proceedings of the 3rd IEEE International Conference on Space Mission Challenges for Information Technology*, Seiten 83–90. IEEE Computer Society, Washington, DC, USA. ISBN 978-0-7695-3637-8. (Zitiert auf den Seiten 27 und 28.)

- [110] Radio Technical Commission for Aeronautics (RTCA) (1992). *DO-178B: Software Considerations in Airborne Systems and Equipment Certification*. (Zitiert auf den Seiten 5 und 17.)
- [111] Reactive Systems, Inc. (2010). *Reactis®*. <http://www.reactive-systems.com>. Letzter Zugriff: Oktober 2010. (Zitiert auf Seite 42.)
- [112] Reactive Systems, Inc. (2010). *Reactis® Model-Based Testing and Validation, User's Guide*. <http://www.reactive-systems.com/reactis/doc/user/index.html>. Letzter Zugriff: Juni 2010. (Zitiert auf Seite 37.)
- [113] I. Rechenberg (1973). *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Fromman-Holzboog, Stuttgart. ISBN 978-3772803741. (Zitiert auf Seite 49.)
- [114] W. Rosenstiel (1999). *Eingebettete Systeme*. In *Automatisierungstechnik, Theoretische Grundlagen, Methoden, Anwendungen*, Band 47, Seiten 283–284. Oldenburg Verlag. (Zitiert auf Seite 3.)
- [115] M. Satpathy, A. Yeolekar und S. Ramesh (2008). *Randomized directed testing (REDIRECT) for Simulink/Stateflow models*. In *Proceedings of the 8th ACM International Conference on Embedded Software*, Seiten 217–226. ACM, New York, NY, USA. ISBN 978-1-60558-468-3. (Zitiert auf den Seiten 30, 45 und 47.)
- [116] H. Schlingloff, C. Sühl, H. Dörr, M. Conrad, J. Stroop, S. Sadeghipour, M. Kühl, F. Rammig und G. Engels (2004). *IMMOS - Eine integrierte Methodik zur modellbasierten Steuergeräte-Entwicklung*. In *BMBF-Statusseminar Software Engineering 2006*. (Zitiert auf Seite 4.)
- [117] K. Sen und G. Agha (2006). *CUTE and jCUTE: Concolic Unit Testing and Explicit Path Model-Checking Tools*. In *Proceedings of the 18th International Conference on Computer Aided Verification*, Seiten 419–423. (Zitiert auf Seite 31.)
- [118] K. Sen, D. Marinov und G. Agha (2005). *CUTE: A Concolic Unit Testing Engine for C*. In *Proceedings of the 10th European Software Engineering Conference held jointly with*

- 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering, Seiten 263–272. ACM, New York, NY, USA. ISBN 1-59593-014-0. (Zitiert auf Seite 30.)
- [119] S. Sims und D. C. DuVarney (2007). *Experience Report: The Reactis Validation Tool*. In Proceedings of the 12th ACM SIGPLAN International Conference on Functional Programming, Seiten 137–140. ACM, New York, NY, USA. ISBN 978-1-59593-815-2. (Zitiert auf den Seiten 4, 42 und 43.)
 - [120] SRI International (2006). *SAL Homepage*. <http://sal.csl.sri.com/>. Letzter Zugriff: August 2010. (Zitiert auf Seite 27.)
 - [121] M. E. Staknis (1990). *Software Quality Assurance through Prototyping and Automated Testing*. Information and Software Technology, 32(1), 26–33. ISSN 0950-5849. (Zitiert auf Seite 23.)
 - [122] T-VEC (2010). *Tester for Simulink and Stateflow*. <http://www.t-vec.com/solutions/simulink.php>. Letzter Zugriff: Oktober 2010. (Zitiert auf Seite 43.)
 - [123] K.-C. Tai (1980). *Program Testing Complexity and Test Criteria*. IEEE Transactions on Software Engineering, 6(6), 531–538. ISSN 0098-5589. (Zitiert auf Seite 4.)
 - [124] B. Tessema und G. G. Yen (2009). *An Adaptive Penalty Formulation for Constrained Evolutionary Optimization*. IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans, 39(3), 565–578. ISSN 1083-4427. (Zitiert auf den Seiten 101, 102 und 103.)
 - [125] The MathWorks™, Inc. (2009). *Model Objects That Receive Model Coverage, Documentation: Simulink Verification and Validation*. <http://www.mathworks.com/help/toolbox/slvnv/ug/bstx5sz.html>. Letzter Zugriff: Mai 2009. (Zitiert auf Seite 127.)
 - [126] The MathWorks™, Inc. (2010). *MATLAB®*. <http://www.mathworks.com/products/matlab>. Letzter Zugriff: Mai 2011. (Zitiert auf den Seiten 4 und 14.)
 - [127] The MathWorks™, Inc. (2010). *Simulink® Design Verifier™*. <http://www.mathworks.com/products/sldesignverifier>. Letzter Zugriff: April 2011. (Zitiert auf Seite 43.)

- [128] The MathWorksTM, Inc. (2010). *Stateflow*[®]. <http://www.mathworks.com/products/stateflow>. Letzter Zugriff: Mai 2011. (Zitiert auf den Seiten 4 und 15.)
- [129] The MathWorksTM, Inc. (2011). *Simulink*[®]. <http://www.mathworks.com/products/simulink>. Letzter Zugriff: Mai 2011. (Zitiert auf den Seiten 4 und 14.)
- [130] The MathWorksTM, Inc. (2011). *Simulink Product Documentation*. <http://www.mathworks.com/help/toolbox/simulink>. Letzter Zugriff: Mai 2011. (Zitiert auf Seite 15.)
- [131] The MathWorksTM, Inc. (2011). *Parallel Computing*. <http://www.mathworks.com/parallel-computing>. Letzter Zugriff: Mai 2011. (Zitiert auf den Seiten 175 und 183.)
- [132] The MathWorksTM, Inc. (2011). *Simulink[®] Demo: Vehicle Electrical and Climate Control Systems*. http://www.mathworks.com/products/simulink/demos.html?file=/products/demos/shipping/simulink/sldemo_auto_climate_elec.html. Letzter Zugriff: April 2011. (Zitiert auf Seite 155.)
- [133] The MathWorksTM, Inc. (2011). *Simulink[®] Demo: Modeling an Automatic Transmission Controller*. http://www.mathworks.com/products/simulink/demos.html?file=/products/demos/shipping/simulink/sldemo_autotrans.html. Letzter Zugriff: April 2011. (Zitiert auf Seite 155.)
- [134] The MathWorksTM, Inc. (2011). *Simulink[®] Demo: Modeling a Fault-Tolerant Fuel Control System*. http://www.mathworks.com/products/simulink/demos.html?file=/products/demos/shipping/simulink/sldemo_fuelsys.html. Letzter Zugriff: April 2011. (Zitiert auf Seite 155.)
- [135] The MathWorksTM, Inc. (2011). *Stateflow Product Documentation*[®]. <http://www.mathworks.com/help/toolbox/stateflow>. Letzter Zugriff: Mai 2011. (Zitiert auf Seite 16.)
- [136] N. Tracey, J. Clark, K. Mander und J. McDermid (1998). *An Automated Framework for Structural Test-Data Generation*. In *Proceedings of the 13th IEEE International Conference on Automated Software Engineering*, Seiten 285–288. (Zitiert auf den Seiten 36 und 41.)

- [137] H. Trauboth (1993). *Software-Qualitätssicherung*. Oldenbourg, München [u.a.]. ISBN 3486206923. (Zitiert auf Seite 17.)
- [138] S. Tripakis, C. Sofronis, P. Caspi und A. Curic (2005). *Translating Discrete-Time Simulink to Lustre*. ACM Transactions on Embedded Computing Systems, 4(4), 779–818. ISSN 1539-9087. (Zitiert auf Seite 28.)
- [139] UK Ministry of Defence (1997). *Defence Standard 00-55: Requirements of Safety Related Software in Defence Equipment*. (Zitiert auf den Seiten 5 und 17.)
- [140] M. Utting und B. Legeard (2006). *Practical Model-Based Testing: A Tools Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. ISBN 0123725011. (Zitiert auf Seite 4.)
- [141] A. Valmari (1998). *The State Explosion Problem*. In Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, Seiten 429–528. Springer-Verlag, London, UK. ISBN 3-540-65306-6. (Zitiert auf Seite 6.)
- [142] M. Y. Vardi und P. Wolper (1986). *An Automata-Theoretic Approach to Automatic Program Verification (Preliminary Report)*. In Proceedings of the 1st Symposium on Logic in Computer Science, Seiten 332–344. IEEE Computer Society. (Zitiert auf Seite 99.)
- [143] W. Visser, C. S. Păsăreanu und S. Khurshid (2004). *Test Input Generation with Java PathFinder*. SIGSOFT Softw. Eng. Notes, 29(4), 97–107. ISSN 0163-5948. (Zitiert auf Seite 28.)
- [144] E. Wallmüller (2001). *Software - Qualitätssicherung in der Praxis*. Hanser Fachbuch. ISBN 3446158464. (Zitiert auf Seite 17.)
- [145] S. Wappler (2007). *Automatic Generation of Object-Oriented Unit Tests Using Genetic Programming*. Dissertation, Technische Universität Berlin. (Zitiert auf den Seiten 34, 36 und 51.)
- [146] J. Wegener, A. Baresel und H. Sthamer (2001). *Evolutionary Test Environment for Automatic Structural Testing*. Information and Software Technology, 43(1), 841–854. (Zitiert auf den Seiten 34, 41, 138, 139, 140 und 147.)

- [147] K. Weierstraß (1885). *Über die analytische Darstellbarkeit sogenannter willkürlicher Funktionen einer reellen Veränderlichen*. Sitzungsberichte der Königlich Preußischen Akademie der Wissenschaften zu Berlin, Seiten 633–639, 789–805. (Zitiert auf den Seiten 69 und 70.)
- [148] B. Wilmes (2010). *Berücksichtigung von Signal-Constraints im suchbasierten Softwaretest kontinuierlicher Systeme*. Diplomarbeit, Technische Universität Berlin. (Zitiert auf Seite 91.)
- [149] B. Wilmes und A. Windisch (2010). *Considering Signal Constraints in Search-Based Testing of Continuous Systems*. In *Proceedings of the 3rd International Conference on Software Testing, Verification, and Validation Workshops*, Seiten 202–211. (Zitiert auf Seite 91.)
- [150] A. Windisch, S. Wappler und J. Wegener (2007). *Applying Particle Swarm Optimization to Software Testing*. In *Proceedings of the 9th annual Conference on Genetic and Evolutionary Computation*, Seiten 1121–1128. (Zitiert auf Seite 82.)
- [151] D. H. Wolpert und W. G. Macready (1997). *No Free Lunch Theorems for Optimization*. *IEEE Transactions on Evolutionary Computation*, **1**(1), 67–82. ISSN 1089-778X. (Zitiert auf Seite 184.)
- [152] M. H. Zaki, S. Tahar und G. Bois (2008). *Formal Verification of Analog and Mixed Signal Designs: A Survey*. *Microelectronics Journal*, **39**(12), 1395–1404. ISSN 0026-2692. (Zitiert auf Seite 93.)
- [153] Y. Zhan (2005). *A Search-Based Framework for Automatic Test-Set Generation for MATLAB/Simulink Models*. Dissertation, University of York. (Zitiert auf den Seiten xv, 37, 39, 125 und 129.)
- [154] Y. Zhan und J. Clark (2004). *Search-Based Automatic Test-Data Generation at an Architectural Level*. In *Proceedings of the 6th Genetic and Evolutionary Computation Conference*, Seiten 1413–1426. (Zitiert auf Seite 37.)
- [155] Y. Zhan und J. A. Clark (2008). *A Search-Based Framework for Automatic Testing of MATLAB/Simulink Models*. *Journal of Systems and Software*, **81**(2), 262–285. ISSN 0164-1212. (Zitiert auf den Seiten 22 und 186.)

EIDESSTATTLICHE ERKLÄRUNG

Die selbständige und eigenhändige Anfertigung dieser Arbeit
versichere ich an Eides statt.

Berlin, November 2011

Andreas Windisch