Automatische Testdatengenerierung mittels multi-objektiver Optimierung

Norbert Oster und Francesca Saglietti
Lehrstuhl für Software Engineering, Universität Erlangen-Nürnberg
http://www11.informatik.uni-erlangen.de
{oster,saglietti}@informatik.uni-erlangen.de

Einführung und Motivation: Um das für eine geplante Anwendung erforderliche Maß an Zuverlässigkeit und Verfügbarkeit zu gewährleisten, repräsentiert die Testphase nach wie vor eine wesentliche Aktivität des Entwicklungsprozesses. Der Hauptgrund für den dabei aufzubringenden zeitlichen und finanziellen Aufwand liegt in dem noch sehr hohen Anteil manuell auszuführender Arbeit – vor allem, wenn nach kontrollfluss- oder datenflussorientierten Strategien verfahren wird. Bereits die Identifizierung der Testfälle zur Erfüllung vorgegebener Testkriterien stellt in der Regel eine komplexe, mühsame und daher fehleranfällige Aufgabe dar. Darüber hinaus müssen nach der Ausführung der ermittelten Testdaten die Ergebnisse der Testläufe hinsichtlich ihrer Erfüllung der Spezifikation bzw. des erwünschten oder erforderlichen Verhaltens analysiert werden. Aus diesem Grunde lohnt es sich, den Validierungsaufwand durch möglichst minimale Testfallmengen zu reduzieren.

Unter Testautomatisierung versteht man heute in der Regel lediglich die automatische Unterstützung der Ausführung vorgegebener Testfälle. Einige Ansätze erlauben darüber hinaus die automatische Testdatengenerierung [WBP02]. Im Vergleich mit den bestehenden Verfahren betrifft die Originalität der im Folgenden vorgestellten Methode [OS06] sowohl den uneingeschränkten Einsatz der Programmiersprache Java (einschließlich rein prozeduraler Programme) als auch die gleichzeitige Verfolgung zweier gegenläufiger Optimierungsziele, nämlich der Maximierung der durch die Testmenge erzielten Überdeckung und der Minimierung der dafür erforderlichen Anzahl an Testfällen. Darüber hinaus eignet sich dieses Verfahren für beliebige strukturelle (einschließlich datenfluss-basierter [RW85]) Teststrategien. Je nach dem geforderten Testkriterium sind verschiedene Entitäten des Kontroll- bzw. Datenflusses zu überdecken. Diese lassen sich vor dem Testen i.A. durch statische Analyse bestimmen. Die Ausführung eines jeden Testfalls muss anschließend dynamisch analysiert werden, um die Anzahl der dadurch überdeckten Entitäten zu ermitteln.

Statische und dynamische Analyse: Auf Basis einer symbolischen Ausführung stellt der statische Analysator jede aufrufbare Methode als datenfluss-annotierten interprozeduralen Kontrollflussgraphen dar. Zur Behandlung von Pointer-Aliases wird im Laufe der symbolischen Ausführung die *points-to*-Menge dynamisch ermittelt [CR01]. Der interprozedurale Kontrollfluss kann sich auf das zu testende System beschränken und externen Code

(z.B. Bibliotheken) ausschließen. Da Schleifen die *points-to*-Menge einiger Variablen beeinflussen können, wurde die dabei eingesetzte Fixpunkt-Iterationsstrategie so optimiert, dass eine wiederholte Analyse von Teilpfaden weitgehend vermieden wird.

Um eine objektive Aussage über die Anzahl von einem Testfall überdeckter Entitäten zu erhalten, wird der Quellcode mit Hilfe von ANTLR (ANother Tool for Language Recognition) instrumentiert, wobei der Code jeder Klasse zunächst in einen abstrakten Syntaxbaum konvertiert wird. Dieser Baum wird anschließend an wohldefinierten Stellen um zusätzliche Knoten ergänzt, die Aufrufe entsprechender Protokoll-Routinen darstellen. Dadurch erhält die Protokollierungseinheit während der Testfallausführung auf effiziente Weise datenflussbezogene Nachrichten vom laufenden System und speichert sie in einem *Laufzeitprotokoll*. Nach der Auswertung der Daten aus dem Laufzeitprotokoll wird der gesamte Datenfluss des zugrunde liegenden Testfalls – basierend auf dem *Instrumentierungsprotokoll* – rekonstruiert. Da jede Instanz einer Klasse zur Laufzeit leicht identifiziert werden kann, wird dadurch ein exaktes *pointer-aliasing* zu niedrigen Kosten erzielt.

Multi-objektive Testdatengenerierung: Um den inhärenten Konflikt zwischen hoher Überdeckung und niedriger Testfallanzahl zu lösen, wird multi-objektive Optimierung eingesetzt. Hierzu wurden vier bekannte Techniken eingesetzt und miteinander verglichen: *Multi-objective Random Search* (RS), *Simulated Annealing* (SA), sowie die Evolutionären Verfahren *Multi-Objective Aggregation* (MOA) und *Non-dominated Sorting Genetic Algorithm* (NSGA) [ZT98]. Die beiden Zielgrößen Überdeckung und Testumfang gehen durch gewichtete Summenbildung (bei RS, SA und MOA) bzw. auf Basis einer sogenannten Dominanzrelation (bei NSGA) in die *Fitness-Funktion* ein, wobei eine Lösung als nicht dominiert gilt, wenn keine weitere bekannte Lösung alle Ziele besser erfüllt. Dadurch bietet NSGA mehrere optimale Ergebnisse entlang der sogenannten Pareto-Front, wodurch die Gratwanderung zwischen den beiden Optimierungszielen deutlich wird [OS06].

Experimentelle Ergebnisse: Zum Zweck der Evaluierung der entwickelten Methode wurden die vier genannten Optimierungsalgorithmen in einem Werkzeug namens •gEAr implementiert, welches in verschiedenen Projekten unterschiedlicher Größe eingesetzt wurde. In den meisten Fällen war SA den Algorithmen MOA und RS überlegen. Erwartungsgemäß scheiterte RS bald an einem lokalen Optimum. Zwar übertraf NSGA das Verfahren SA nicht in Bezug auf die Leistungsfähigkeit, bietet dafür aber mehr Flexibilität, weil er nicht nur ein einzelnes Optimum liefert, sondern die ganze optimale Front. Insgesamt ergaben die metaheuristischen Ansätze eine deutliche Verbesserung gegenüber klassischen zufallsbasierten Suchverfahren. Zwecks Untersuchung des Fehleraufdeckungspotentials automatisch generierter Testdaten minimierten Umfangs wurde als Indikator der sogenannte *mutation score* verwendet.

Für zwei Überdeckungskriterien wurde das Verfahren eingesetzt, wobei Testumfang und Überdeckungsgrad im Verhältnis 0,05:1 gewichtet wurden. Die in Tabelle 1 angegebene Schätzung der prozentualen Überdeckung ist als konservativ zu betrachten, da sie auf allen statisch ermittelbaren Entitäten beruht, ohne zu differenzieren, ob diese während der Ausführung tatsächlich überdeckt werden können. Um den Anteil äquivalenter Mutanten abschätzen zu können, wurde statistische Stichprobentheorie auf eine große Anzahl zufällig

		Verzweigungsüberdeckung				all-uses				Mutationstesten		
Projekt	NM	GT	CE	COV	MS	GT	CE	COV	MS	ET	RT	MS
BigFloat	1528	5	144	98,6	69,2	17	1511	82,8*	75,9	3000	232	95,8
Dijkstra	220	1	26	100	70,5	2	168	93,8*	71,8	10000	8	75,9
Hanoi	227	2	4	100	74,9	2	42	96,7	76,7	1000	11	85,9
Huffman	623	3	61	100	74,3	3	353	92,9*	75,4	n/a	6	100
JDK sort	852	1	37	100	61,5	2	315	83,4	65,4	4000	108	81,5
JDK log	1970	35	317	65,2	24,6	61	1599	82,8*	25,2	n/a	n/a	n/a

Tabelle 1: Experimentelle Ergebnisse

NM: Gesamtanzahl generierter Mutanten (einschließlich funktional äquivalenter Mutanten)
GT: Anzahl generierter Testfälle
CE: Anzahl überdeckter Entitäten (Verzweigungen bzw. def/use-Paare)
COV: Überdeckung (in %; aufgrund der statischen Analysis)
MS: Verhältnis zwischen getöteten Mutanten und NM (in %)
ET: Gesamtzahl gleich-verteilter Testfälle beim Mutationstesten
RT: Anzahl effektiver Testfälle beim Mutationstesten
* Durchschnitt über die analysierbaren Methoden

generierter Eingaben angewendet. Daraus kann mit hoher Aussagesicherheit geschlossen werden, dass überlebende Mutanten zum ursprünglichen Programm funktional äquivalent sind. In Anbetracht dieser Überlegungen zeigen die Ergebnisse in Tabelle 1 deutlich, dass das hierbei erzielte Fehlererkennungsvermögen um mehrere Größenordnungen besser als bei klassisch zufallsgenerierten Testfällen ist (Einzelheiten in [OS06]).

Schlussfolgerung und Ausblick: Die bisherigen Erfahrungen bestätigen Effizienz und Anwenderfreundlichkeit der entwickelten Methode. Deren Leistungsfähigkeit ließe sich etwa durch *Hybridisierung* (z.B. als Kombination mit lokaler Optimierung) weiter verbessern. Im Rahmen des öffentlich geförderten Verbundprojektes *UnITeD* mit zwei industriellen Partnern wird derzeit der beschriebene Testprozess auf die frühen Phasen modellgetriebener Software-Entwicklung erweitert und im medizinischen Bereich erprobt.

Literatur

[CR01] R. Chatterjee und B. G. Ryder. Data-flow-based Testing of Object-Oriented Libraries. Bericht, Department of Computer Science, Rutgers University, 2001.

[OS06] N. Oster und F. Saglietti. Automatic Test Data Generation by Multi-Objective Optimisation. In *Computer Safety, Reliability and Security*, LNCS. Springer-Verlag, 2006.

[RW85] S. Rapps und E. J. Weyuker. Selecting Software Test Data Using Data Flow Information. *IEEE Transactions on Software Engineering*, 1985.

[WBP02] J. Wegener, K. Buhr und H. Pohlheim. Automatic Test Data Generation For Structural Testing Of Embedded Software Systems By Evolutionary Testing. In *Genetic and Evolutionary Computation Conference*. Morgan Kaufmann Publishers, 2002.

[ZT98] E. Zitzler und L. Thiele. Multiobjective Optimization Using Evolutionary Algorithms – A Comparative Case Study. Bericht, Swiss Federal Institute of Technology Zurich, 1998.