Strukturorientiertes Testen funktionaler Programme

Manfred Widera

Kurzfassung

1 Einleitung

Funktionale Programmierung ermöglicht das Erstellen von Programmen auf einem hohen Abstraktionsniveau. Eine Anzahl erfolgreicher Softwareprojekte unter Verwendung funktionaler Sprachen lässt eine wachsende Relevanz funktionaler Programmierung für die industrielle Softwareentwicklung erwarten.

Gerade für die industrielle Nutzung einer Programmiersprache ist jedoch eine ausreichende Werkzeugunterstützung notwendig. Die Verfügbarkeit solcher Werkzeuge für funktionale Sprachen ist bisher noch nicht ausreichend, insbesondere auch was das Testen funktionaler Programme betrifft. Speziell das strukturorientierte Testen funktionaler Programme ist bisher kaum berücksichtigt worden.

Bei dem Versuch, Ansätze zum strukturorientierten Testen von der imperativen Programmierung auf funktionale Programme zu übertragen, treten zwei Hauptprobleme auf: Funktionen höherer Ordnung komplizieren die Erstellung von Flussgraphen, und die Verwendung komplexer kontrollflussorientierter Überdeckungskriterien scheitert daran, dass die dazu passenden Programmkonstrukte in funktionalen Programmen kaum vorkommen.

Zur Flussgrapherstellung wird hier ein Ansatz vorgestellt, der durch iterierte Datenflussanalyse die Ziele von Funktionsaufrufen höherer Ordnung berechnet. Dieser Ansatz basiert auf dem 0CFA-Verfahren, das in Übersetzern für funktionale Sprachen zum Einsatz kommt. Die Flussgraphen für das strukturorientierte Testen müssen insbesondere für die Programmierer in der jeweiligen Sprache verständlich sein und eine möglichst große Ähnlichkeit mit dem repräsentierten Code aufweisen. Dazu wird ein spezielles Konzept bidirektionaler Aufrufkanten eingeführt.

Für den Überdeckungstest in der funktionalen Programmierung bieten sich datenflussorientierte Überdeckungskriterien an. Diese sind weitgehend unabhängig von den verwendeten Programmkonstrukten und können auch für funktionale Sprachen mit verzögerter Auswertung sinnvoll eingesetzt werden. Sequenzen von Definitions-Verwendungs-Paaren erlauben es, den Datenfluss von Werten losgelöst von den transportierenden Variablen zu betrachten.

2 Funktionale Flussgraphen

Die meisten Überdeckungskriterien für das strukturorientierte Testen sind auf der Basis von Flussgraphen definiert. Diese Flussgraphen müssen für die mit dem Testen beauftragten Programmierer/Tester einfach verständlich sein. Daraus ergibt sich die Anforderung, bei der Präsentation der Flussgraphen eine möglichst hohe Ähnlichkeit zum repräsentierten Quellcode zu wahren.

Funktionsaufrufe haben eine hohe Bedeutung in funktionalen Programmiersprachen. Insbesondere werden Schleifen vollständig durch Rekursion und damit durch Funktionsaufrufe ausgedrückt. Für das strukturorientierte Testen funktionaler Programme kommen daher nur interprozedurale Flussgraphen in Frage. Der Repräsentation von Funktionsaufrufen muss besondere Aufmerksamkeit geschenkt werden.

Die vorgestellten funktionalen Flussgraphen verwenden spezielle Aufrufkanten. Diese sind bidirektional und repräsentieren sowohl den Funktionsaufruf, als auch die Rückkehr von diesem Aufruf. Ein Aufrufknoten im Flussgraphen verfügt nun im allgemeinen über mehrere ausgehende Kanten:

- Null, ein oder mehrere Aufrufkanten verlaufen von dem Aufrufknoten zu den Startknoten aller potentiell aufgerufenen Funktionen innerhalb des Flussgraphen.
- Eine normale Kante repräsentiert den Kontroll- und Datenfluss nach der Rückkehr vom Funktionsaufruf.

Durch diese Darstellung wird die Notwendigkeit für explizite Rückkehrkanten umgangen. Dies vermeidet nicht verfolgbare Pfade, d.h. Pfade bei denen Aufrufkanten und Rückkehrkanten nicht in korrekter Paarung auftreten. Darüber hinaus erlaubt die Unterscheidung der Kantentypen, von den Funktionsaufrufen zu abstrahieren und lediglich den Kontroll-/Datenfluss innerhalb einer Funktion zu betrachten, ohne den Flussgraphen dazu anpassen zu müssen.

Ein generelles Problem bei der Berechnung von Flussgraphen für funktionale Programme ist durch evtl. vorhandene Funktionen höherer Ordnung gegeben. Funktionsaufrufe können an der Position der aufgerufenen Funktion einen beliebigen Ausdruck enthalten. Um die möglichen Ziele eines Funktionsaufrufs bereits zur Übersetzungszeit zu bestimmen, bietet sich die Datenflussanalyse an.

Datenflussanalyseverfahren sind bereits in Übersetzern für funktionale Programmiersprachen zu finden. Für die Bestimmung von Flussgraphen für das

strukturorientierte Testen funktionaler Programme hat sich ein Ansatz basierend auf dem bekannten 0CFA-Verfahren bewährt.

Eine Betrachtung des Datenflusses einer *Variablen* von ihrer Definition zur Verwendung stellt sich jedoch als nicht ausreichend heraus. Vielmehr ist es sinnvoll, den Datenfluss eines *Wertes* von seiner Definition zur Verwendung zu verfolgen. Dazu ist eine Reihe von Datenflussformen zu berücksichtigen, die den Datenfluss von Werten über die Grenzen einer Variablenbindung hinweg repräsentieren:

- Binden von Werten an neue Variablen. Dies kann explizit durch entsprechende Matchings geschehen oder implizit bei der Parameterübergabe von Funktionsaufrufen.
- Binden von Rückgabewerten von Funktionen.
- Verwendung von Bindungen aus dem Definitionskontext eines Lambdaabschlusses.
- Speicherung eines Wertes in einer Struktur (Cons-Zelle, Tupel, ...) und spätere Selektion aus dieser Struktur.

Diese Formen von Datenfluss erweisen sich als gut geeignet zur Repräsentation des Datenflusses in funktionalen Programmen.

3 Überdeckungskriterien

Strukturorientiertes Testen kann auf der Basis unterschiedlicher Überdeckungskriterien erfolgen. Bei den bekannten Ansätze werden üblicher Weise kontrollflussorientierte und datenflussorientierte Überdeckungskriterien unterschieden.

3.1 Kontrollfluss-Überdeckung

Für die Anwendung der bekannten kontrollflussorientierten Überdeckungskriterien auf funktionale Programme ergibt sich ein zweigeteiltes Bild: Während die einfachen Kriterien (Knotenüberdeckung und Kantenüberdeckung) direkt auf funktionale Programmiersprachen übertragbar sind, ergeben sich für die komplexeren Verfahren diverse Probleme, da sich die Kontrollflussstrukturen funktionaler Programmiersprachen deutlich von denen imperativer Sprachen unterscheiden:

- Schleifenkonstrukte kommen in funktionalen Programmiersprachen nicht vor. Schleifen werden durch Funktionsaufrufe und Rekursion gebildet.
- Verzweigungen basieren in modernen funktionalen Programmiersprachen auf der Mustererkennung. Komplexe Prädikate, die segmentweise getestet werden könnten, kommen in funktionalen Programmen kaum vor.

Ein weiterer allgemeiner Nachteil kontrollflussorientierter Überdeckungskriterien ergibt sich bei der Betrachtung funktionaler Programmiersprachen mit verzögerter Auswertung. Der Kontrollfluss in einem Programm ist hier selbst für den Programmierer nicht mehr einfach zu verstehen und vorherzusehen.

3.2 Datenfluss-Überdeckung

Die Betrachtung der datenflussorientierten Überdeckung ist für funktionale Programme naheliegend und sinnvoll. Funktionsaufrufe lassen sich gut durch den Datenfluss der Argumente beim Aufruf und des Berechnungsergebnisses bei der Rückkehr beschreiben. Darüber hinaus ist der Datenfluss auch in Programmiersprachen mit verzögerter Auswertung gut zu überblicken.

Die für die Flussgrapherstellung verwendeten Formen von Datenfluss dienen als Basis für die Definition von datenflussorientierten Überdeckungskriterien. Dies geschieht durch die Definition von *Definitions-Verwendungs-Ketten*, d.h. Sequenzen von (in üblicher Weise definierten) Definitions-Verwendungs-Paaren mit der Eigenschaft, dass die Definition des ersten Paares und die Verwendung des letzten Paares einer Kette den gleichen Wert bezeichnen.

Da die Anzahl der betrachteten Definitions-Verwendungs-Ketten beim Überdeckungstest deutlich höher ist als bei der Flussgrapherstellung, kann bei manchen Programmen eine Einschränkung der betrachteten Datenflussformen notwendig sein.

4 Zusammenfassung

Für funktionale Programmiersprachen ist eine höhere Verfügbarkeit von Werkzeugen anzustreben. Die hier vorgestellten Konzepte zielen auf die Bereitstellung des strukturorientierten Testens für funktionale Programme ab. Dazu wurden Ansätze für die Flussgrapherstellung und für den Überdeckungstest vorgestellt.

Bei der Erstellung und Präsentation der Flussgraphen ist eine möglichst große Nähe zum repräsentierten Programmcode anzustreben. Dazu wurden bidirektionale Aufrufkanten eingeführt. Durch sie ist es möglich, sowohl den lokalen Kontroll- und Datenfluss innerhalb einer Funktion, als auch den globalen Fluss durch Funktionsaufrufe mit Hilfe der gleichen Struktur darzustellen.

Bei der Auswahl der Überdeckungskriterien zeigen sich insbesondere die datenflussorientierten Kriterien als geeignet. Sie beschreiben die in funktionalen Sprachen typischen Kontrollstrukturen sinnvoll und erlauben auch die Betrachtung funktionaler Sprachen mit verzögerter Auswertung.

Insgesamt erlauben die vorgestellten Konzepte die Anwendung des strukturorientierten Testens auf funktionale Programmiersprachen und eliminieren damit eine Hürde auf dem Weg dieser Sprachen zur breiten industriellen Anwendung.