Systematisches Testen Logisch-Funktionaler Programme

Sebastian Fischer
Institut für Informatik
Christian-Albrechts-Universität zu Kiel

Herbert Kuchen
Institut für Wirtschaftsinformatik
Westfälische Wilhelms-Universität Münster

Logisch-funktionalen Programmiersprachen liegt der Auswertungs-Mechanismus *Narrowing* zugrunde – eine verallgemeinerte *Reduktion*, bei der statt Musteranpassung auch so genannte *freie* Variablen gebunden werden können. Dadurch ist es möglich, Funktionen mit unbekannten Werten aufzurufen, die während der Berechnung instantiiert werden.

Wir verwenden diesen Mechanismus zur Testfallgenerierung im Kontext der logisch-funktionalen Programmiersprache Curry (Hanus et al. 2006). Dabei werden Testfälle durch Narrowing erzeugt und anhand eines variablen Quelltext-Überdeckungskriteriums ausgewählt.

Wir präsentieren zwei unterschiedliche Überdeckungskriterien und übertragen dabei die Idee der Kontrollfluss-Überdeckung – bekannt aus dem Umfeld imperativer Programmiersprachen – auf eine deklarative Sprache.

Andere Werkzeuge zum Testen funktionaler Programme (Claessen and Hughes 2000; Koopman et al. 2002) generieren Testfälle anhand der Schnittstelle (Typsignatur) der zu testenden Funktionen. Diesen Ansatz nennt man *Black-Box-Testing*, da die Implementierung der zu testenden Funktion ignoriert wird. Im Kontrast dazu bezeichnet man einen Ansatz, der die konkrete Implementierung einer Funktion berücksichtigt, als *White-* oder treffender als *Glass-Box-Testing*.

Unser Werkzeug greift eine Arbeit von Müller et al. (2004) auf und ist das erste, dass *Glass-Box-Testing* deklarativer Programme unterstützt. Es basiert auf einer Transformation, die ein Curry Programm so anreichert, dass es bei seiner Ausführung Informationen zur Quelltext-Überdeckung protokolliert. Dies ist in Programmiersprachen mit bedarfsgesteuerter Auswertung eine besondere Herausforderung. Interessanterweise, lässt sich diese meistern, ohne nicht-deklarative Operationen zu verwenden.

Eine prototypische Implementierung verdeutlicht den Wert unseres Ansatzes für die Entwicklung zuverlässiger Software. Überdeckende Mengen von Testfällen für eine mittelgroße Arithmetik-Bibliothek können innerhalb weniger Sekunden berechnet werden. Überdeckende Testfälle von Hand zu erzeugen wäre aufgrund der Komplexität der Algorithmen – wenn überhaupt – nur mit hohem Aufwand möglich.

Literatur

Koen Claessen and John Hughes. QuickCheck: a lightweight tool for random testing of Haskell programs. *ACM SIGPLAN Notices*, 35(9):268–279, September 2000. URL `http://www.md.chalmers.se/~rjmh/QuickCheck/`.

M. Hanus et al. Curry: An integrated functional logic language (version 0.8.2). Available at URL `http://www.informatik.uni-kiel.de/~curry`, 2006.

Pieter Koopman, Artem Alimarine, Jan Tretmans, and Rinus Plasmeijer. Gast: Generic automated software testing. In R. Peña, editor, *The 14th International workshop on the Implementation of Functional Languages, IFL'02, Selected Papers*, volume 2670 of *LNCS*, pages 84–100. Madrid, Spain, Springer, September 2002.

R. Müller, C. Lembeck, and H. Kuchen. A symbolic java virtual machine for test-case generation. In *Proceedings IASTED*, 2004.