Stackless Stack Inspection
Portabler Fluchtweg aus dem Teufelskreis

Baltasar Trancón y Widemann
Universität Bayreuth

Zusammenfassung

Tritt bei der rekursiven Auswertung eines Funktion ein Zyklus auf, dann führt das im Allgemeinen zu einer nichtterminierenden Berechnung, beziehungsweise zum Programmabbruch wegen Überlauf des Aufrufstacks. Semantisch betrachtet wären aber viele Funktionen in diesem Fall trotzdem wohldefiniert, zum Beispiel primitiv corekursive Funktionen oder Prädikate mit Fixpunktsemantik. Damit die Berechnung effektiv zu Ende geführt werden kann, muss der Zyklus zunächst erkannt, und anschließend behandelt werden. Ersteres kann durch die Inspektion des Aufrufstacks zur Laufzeit erreicht werden; letzteres durch eine geeignete Aufrufkonvention, die es erlaubt, Funktionsergebnisse vor dem rekursiven Unteraufruf teilweise zu fixieren und im Zyklusfall darauf zurückzugreifen. Während die Zyklenbehandlung für die erwähnten Problemklassen leicht durch einfache, portable Operationen implementiert werden kann, ist die vorherige Zyklenerkennung eher problematisch: Maschinennahe Implementierungssprachen (C, C++, C--, etc.) erlauben meist nicht, in portabler Weise auf Teile des Aufrufstacks zuzugreifen; Codeoptimierungen, die den Inhalt des Stacks beeinflussen, erschweren die Realisierung zusätzlich. Virtuelle Maschinen (JVM, CLR, etc.) unterstützen dagegen prinzipiell die Inspektion des Aufrufstacks, stellen aber aus Sicherheitsgründen meist nur die Aufrufstelle, nicht aber die ebenso dringend benötigten Argumentwerte zur Verfügung, da Stackinspektion traditionell nur der Sicherheitskontrolle dient.

Im Zusammenhang mit *Continuations* als Kontrollflusskonstrukt sind Sprachimplementierungen untersucht worden, die den Aufrufstack ganz oder teilweise durch eine explizit verwaltete, verkettete Datenstruktur ersetzen; solche Implementierungen werden als *stackless* bezeichnet. Hier ist besonders *Stackless Python* zu erwähnen. Die Technik scheint vielversprechend, um Zyklenerkennung portabel in einer maschinennahen Implementierungssprache zu beschreiben. Besonders attraktiv ist ein hybrider Ansatz, bei dem grundsätzlich der Aufrufstack der physischen Maschine verwendet wird, der nur bei Bedarf in wohldefinierte Datenstrukturen „abgewickelt“ wird. Dadurch bleiben Optimierungen der Implementierungssprache, die den Aufrufstack betreffen, effektiv.

Am Beispiel einer einfachen, wohlbekannten Funktion auf Listen soll die Erweiterung des Definitionsbereichs auf zyklische Daten und die prinzipielle Implementierung in C++ demonstriert werden. Dann wird gezeigt, wie die *stackless*-Zyklenerkennung zu realisieren ist. Dabei treten wegen der speziellen Aufrufkonventionen Probleme auf, die auf trickreiche Art, insbesondere durch Einsatz von Zeigern vierten und fünften Grades, gelöst werden.