Flexibler Abbruch von Anweisungen in LISP TeriPI

Christian Heinlein

Hochschule Aalen – Technik und Wirtschaft
christian.heinlein@hs-aalen.de

Erweiterte Zusammenfassung

Viele imperative Programmiersprachen bieten Anweisungen, mit denen umschließende Anweisungen vorzeitig beendet werden können, zum Beispiel (Syntax gemäß C, Java u. ä.):

- `continue` bricht den aktuellen Durchlauf der umschließenden Schleife ab und beginnt ggf. sofort mit dem nächsten Durchlauf (sofern die Schleifenbedingung noch erfüllt ist).
- `break` bricht die gesamte umschließende Schleife ab.
- `break` mit einer Marke (label) bricht in Java die umschließende Anweisung ab, die mit dieser Marke gekennzeichnet ist. (Dabei muss es sich nicht unbedingt um eine Schleife handeln. Obwohl dies stark an `goto` in C erinnert, sind damit nur Sprünge von innen nach außen und somit kein beliebiger „Spaghetti-Code“ möglich.)
- `return` bricht die umschließende Prozedur (Funktion, Methode, Konstruktor o. ä.) ab. (Darüber hinaus kann `return` einen Resultatwert zurückgeben; dieser Aspekt wird hier jedoch nicht betrachtet.)
- `throw` bricht eine unbestimmte Anzahl umschließender Anweisungen und ggf. Prozeduren ab, bis die von dieser Anweisung geworfene Ausnahme mittels `try-catch` wieder aufgefangen wird.

Damit ist `throw` offensichtlich das allgemeinste und „stärkste“ Konstrukt dieser Art, sodass sich die anderen prinzipiell mit dessen Hilfe simulieren lassen. Um beispielsweise eine Schleife vorzeitig beenden zu können, kann man diese in eine `try-catch`-Anweisung einbetten und dann den Abbruch mittels `throw` realisieren (Syntax gemäß C++):

```
// Leere Dummy-Klasse zur Verwendung als Ausnahme.
class Break {};

try {
    // Schleife.
    for (int i = 1; i <= 10; i++) {
        ......
```

```
            // Vorzeitiger Abbruch bei i == 5.
            if (i == 5) throw Break();
            ......
        }
    }
    catch (Break) {
        // Leerer catch-Block.
    }
```

Tatsächlich wird in der Sprachdefinition von Modula-3 die Semantik von `EXIT` und `RETURN` (wobei `EXIT` dem `break` in C-artigen Sprachen entspricht) auf die von `RAISE` (entspricht `throw`) zurückgeführt.

Obwohl derartige „konzeptuelle Sparsamkeit“ für die formale Beschreibung einer Programmiersprache angebracht sein mag, würde sie beim praktischen Einsatz auf Kosten des Benutzers gehen, d. h. dieser möchte neben dem allgemeinsten Konstrukt `throw` auch die spezielleren Anweisungen wie `break` und `return` nicht missen.

In einer syntaktisch erweiterbaren Programmiersprache wie MOSTflexiPL (vgl. http://flexipl.info) verhält sich dies jedoch anders: Hier stehen konzeptuelle Sparsamkeit und Benutzerfreundlichkeit nicht im Widerspruch zueinander, weil sich bequem verwendbare Spezialkonstrukte jederzeit leicht unter Verwendung eines allgemeineren Mechanismus definieren lassen. Tatsächlich ist konzeptuelle Sparsamkeit eines der Grundprinzipien dieser Sprache:

- Es gibt beispielsweise nur eine einzige vordefinierte Fallunterscheidung, mit deren Hilfe weitere Verzweigungsarten wie z. B. `switch-case` definiert werden können.
- Ebenso gibt es nur einen einzigen vordefinierten Schleifentyp, mit dessen Hilfe sich beliebige weitere – kopfgesteuert, fußgesteuert, Zählschleife, Mengenschleife etc. – leicht definieren lassen.
- Es gibt nur eine „Handvoll“ Grundkonstrukte für parallele Programmierung, mit deren Hilfe sich zahllose Spezialkonstrukte wie z. B. Monitore, Futures, Pipes, MVars etc. realisieren lassen.
- Usw.

Dementsprechend gibt es in MOSTflexiPL auch nur eine einzige allgemeine Sprung- bzw. Abbruchanweisung, die ähnlich mächtig ist wie `throw`, tatsächlich aber sogar noch weitere Anwendungsmöglichkeiten bietet:

- Wenn ein Zweig einer parallelen Ausführung `...||...` eine Anweisung abbricht, die die parallele Ausführung umschließt, muss dadurch zwangsläufig auch der andere Zweig abgebrochen werden.
  Anders als bei den bisher betrachteten Fällen, erfolgt der Abbruch dieses anderen Zweigs dann nicht von „innen“, sondern von „außen“.
- Die Marken, mit denen abbrechbare Anweisungen gekennzeichnet werden können, sind „Bürger erster Klasse“, die z. B. als Parameter an Hilfsfunktionen übergeben oder in Variablen gespeichert werden können.

- Damit muss sich eine Anweisung wie `break` zur vorzeitigen Beendigung einer Schleife nicht mehr zwingend im lexikalischen Gültigkeitsbereich der Schleife befinden, sondern kann – wie jeder andere Code, der innerhalb der Schleife ausgeführt werden soll – auch in eine Hilfsfunktion verlagert werden, was in gängigen Programmiersprachen nicht möglich ist. (Wenn man `break` dort allerdings mittels `throw` simuliert, ist es interessanterweise doch möglich!)
- Wenn die Marke einer Anweisung in einer geeigneten Variablen gespeichert wird, kann eine parallel ausgeführte Anweisung diese ebenfalls benutzen, um die markierte Anweisung von „außen“ abzubrechen.
- Außerdem kann es natürlich vorkommen, dass eine Anweisung abgebrochen werden soll, obwohl sie bereits beendet ist. In diesem Fall ist die Abbruchanweisung einfach wirkungslos.

Aufgrund der vielfältigen Verwendungsmöglichkeiten, insbesondere im Zusammenspiel mit parallelen Ausführungen, ist die implementierungstechnische Umsetzung der beschriebenen Abbruchanweisung relativ komplex und erfordert sorgfältige Synchronisation, um fehlerhafte Ausführungen (race conditions) einerseits und Verklemmungen andererseits zu vermeiden.

Um Abbrüche von außen überhaupt portabel (d. h. ohne direkte Verwendung spezieller Betriebssystemmechanismen) zu ermöglichen, wird kooperatives Multitasking verwendet: Jeder Thread des MOSTflexiPL-Laufzeitsystems überprüft regelmäßig einen ihm zugeordneten Abbruchindikator, der von außen, d. h. von einem anderen Thread, gesetzt werden kann. Wenn ein Thread gerade stillsteht, weil er z. B. auf eine Benutzereingabe oder die Freigabe eines Semaphors wartet, sind diese Blockierungszustände durch geeignete Vorkehrungen so gestaltet, dass sie ebenfalls jederzeit von außen unterbrochen werden können.

Auf diese Weise kann das gesamte MOSTflexiPL-Laufzeitsystem ausschließlich unter Verwendung von Mitteln der C++11-Standardbibliothek realisiert werden.