Parallele Programmierung mit MOSTflexiPL

Christian Heinlein

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

Extended Abstract

MOSTflexiPL (Modular, Statically Typed, Flexibly Extensible Programming Language, siehe http:// flexipl.info) ist eine Programmiersprache, deren Syntax vom Anwender nahezu beliebig erweitert und angepasst werden kann. Aufbauend auf einer kleinen Menge vordefinierter Grundoperatoren, können nach Belieben weitere Operatoren für unterschiedlichste Zwecke definiert werden. Da Operatoren beliebig viele Namen besitzen und auf beliebig viele Operanden angewandt werden können, decken sie neben den üblichen Präfix-, Infix- und Postfix-Operatoren (z. B. `-x` für Vorzeichenwechsel, `x+y` für Addition oder `n!` für Fakultät; die Operatornamen sind jeweils fett gedruckt) auch Mixfix-Operatoren (z. B. `a[i]` für Array-Elementzugriff), Kontrollstrukturen (z. B. **`if`** `x` **`then`** `y` **`else`** `z` **`end`**), Typkonstruktoren (z. B. **`List`** `T`) und Deklarationsformen (z. B. `S` **`->`** `T`, um `S` als Untertyp von `T` zu deklarieren) ab.

Darüber hinaus gibt es einige wenige Grundoperatoren für parallele Programmierung:

- `x||y` führt die Teilausdrücke `x` und `y` parallel aus und wartet auf die Beendigung von beiden, während `||x` den Teilausdruck `x` asynchron ausführt, ohne auf seine Beendigung zu warten.
- `s : sem` deklariert ein Semaphor `s` mit Anfangswert 0.
  `s!+` bzw. `s!-` entsprechen den Operationen V und P von Dijkstra, d. h. sie erhöhen bzw. erniedrigen das Semaphor `s` um 1, wobei `s!-` anschließend ggf. wartet, bis der Wert des Semaphors wieder mindestens 0 ist.

Aufbauend auf diesen Grundoperatoren, kann jeder Programmierer ohne große Mühe „höherwertige“ Operatoren für parallele Programmierung realisieren, zum Beispiel:

- Eine `for`-Schleife, deren Schleifenrumpf für jeden Wert der Laufvariablen parallel ausgeführt wird
- Variablen, die immer abwechselnd geschrieben und gelesen werden müssen (sog. MVars in Haskell)
- Asynchron berechnete Werte (sog. futures), Monitore u. v. a. m.

Damit eignet sich MOSTflexiPL auch hervorragend zum Experimentieren mit neuen Sprachkonstrukten, weil sich diese häufig in wenigen Minuten implementieren lassen.

Eine weitere nützliche Grundoperation, die zunächst nichts mit paralleler Programmierung zu tun hat, ist `abort`, um Ausführungen vorzeitig abzubrechen. Im folgenden Beispiel wird ein Array `a` der Größe `N` nach einem Wert `x` durchsucht. Sobald der Wert gefunden wurde, wird die `for`-Schleife, die mit der Marke `search` gekennzeichnet ist, mittels `abort search` abgebrochen:

```
search ::
for i = 1 .. N do
    if a[i] = x then
        abort search
    end
end
```

Allerdings gibt es zwischen `abort` und der parallelen Ausführung `...||...` eine interessante Wechselwirkung, die im folgenden Beispiel illustriert wird:

```
search :: (
    for i = 1 .. N/2 do
        if a[i] = x then
            abort search
        end
    end
||
    for i = N/2 + 1 .. N do
        if a[i] = x then
            abort search
        end
    end
)
```

Um die Suche nach dem Wert `x` zu beschleunigen, werden die beiden Hälften des Arrays parallel durchlaufen. Sobald der Wert in einer Hälfte gefunden wurde, soll die gesamte Suche abgebrochen werden. Deshalb führt ein `abort` in einem Zweig einer parallelen Ausführung auch zum Abbruch des jeweils anderen Zweigs.

Das folgende Beispiel zeigt eine weitere nützliche Anwendung dieses Prinzips:

```
search :: ((
    for i = 1 .. N do
        if a[i] = x then
            abort search
        end
    end;
    abort search
) || (
    sleep 2000;
    abort search
))
```

Die Suche nach dem Wert `x` soll nach spätestens 2000 Millisekunden abgebrochen werden. Hierfür wird parallel zur Suche eine entsprechend lange `sleep`-Operation ausgeführt, nach deren Beendigung `abort search` ausgeführt wird, wodurch auch die Schleife im anderen Zweig der parallelen Ausführung abgebrochen wird. Wenn die Schleife bereits früher beendet ist, wird anschließend ebenfalls `abort search` ausgeführt, wodurch umgekehrt auch die `sleep`-Operation im anderen Zweig abgebrochen wird.

Mit dem gleichen Mechanismus ist es auch möglich, Operationen vorzeitig abzubrechen, die prinzipiell „endlos“ lange dauern können, z. B. das Warten auf eine Benutzereingabe oder das Warten auf die Freigabe eines Semaphors. Das folgende Beispiel zeigt einen allgemein verwendbaren Operator `try ... for ... ms`, der als Operanden eine beliebige (unausgewertete) Aktion `action` (mit beliebigem Typ `T`) und eine maximale Wartezeit `time` in Millisekunden erhält:

```
try [<T:type>] <{action:T}> for <time:int> ms : bool
{
    wait :: ((
        action;
        abort wait
    ) || (
        sleep time;
        abort wait
    ))
}
```

Eine Anwendung dieses Operators wie z. B. `try s!- for 2000 ms` führt dann die Aktion `action` (im Beispiel `s!-`) und die Operation `sleep time` (im Beispiel `sleep 2000`) parallel aus. Sobald eine der beiden Ausführungen beendet ist, wird der gesamte Block, der mit `wait` gekennzeichnet ist, abgebrochen, woraus folgt, dass die Ausführung von `action` nach spätestens `time` Millisekunden abgebrochen wird.

Dieses Beispiel zeigt erneut das zentrale Grundprinzip von MOSTflexiPL: Aufbauend auf einer kleinen Menge essentieller Grundoperationen, kann jeder Programmierer ohne große Mühe nach Belieben komplexere Operationen definieren. Wenn bestimmte solcher Operationen häufiger benötigt werden, kann man sie in einer Bibliothek aufbewahren, die dann in unterschiedlichen Programmen verwendet werden kann.

Implementierungstechnisch birgt das Zusammenspiel zwischen `abort` und parallelen sowie asynchronen Ausführungen einige Herausforderungen, die durch sorgfältige Analysen der möglichen Wechselwirkungen und semiformale Korrektheitsüberlegungen gemeistert wurden. Damit blockierende Operationen wie `sleep`, `read` und `...!-` durch `abort` abgebrochen werden können, ohne hierfür systemabhängige Spezialmechanismen zu verwenden, mussten sie stellenweise „unkonventionell“ implementiert werden. Auf diese Weise konnte das gesamte MOSTflexiPL-Laufzeitsystem vollkommen portabel in C++11 implementiert werden.