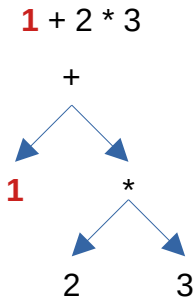
1) Theorie

Schauen wir uns das mal an am Beispiel der Rechnung ***1 + 2 * 3*** wie das konvertieren eines Baumes in eine ***Umgekehrte Polnische Notation*** funktioniert:

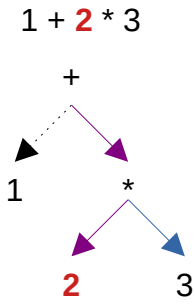
Zuerst wird von der Wurzel ausgehend in nach links verzweigt. Dort verweilt „nur“ die 1 und diese wird als erste auf dem Stack geschrieben. Da die 1 ein Endknoten ist, kann von dort nicht weiter verzweigt werden. Also kehrt der Algorithmus zurück zu der Wurzel (Abbildung 1). Da der komplette linke Ast abgearbeitet worden ist, kehrt der Algorithmus zurück zu der Wurzel, um anschließend den rechten Ast abzuarbeiten.



1

Abbildung 1: Erster Schritt

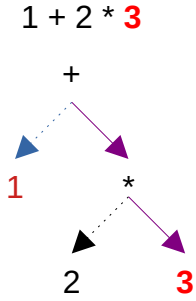
In dem rechten Ast kommt zuerst die Multiplikation vor. Da zuerst die Endknoten und erst anschließend deren (lokale) Wurzel Bearbeitet werden verzweigt der Algorithmus wiederum nach links. Dort befindet sich mit der „2“ ein weiterer Endknoten. Also wird dieser auf dem Steck geschoben (Abbildung 2).



2
1

Abbildung 2: Schritt zwei

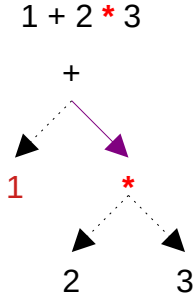
Nun kehrt der Algorithmus zu seiner (lokalen) Wurzel zurück, und verzweigt dieses mal nach rechts. Auch dort befindet sich ein Endknoten („3“). So das dieser jetzt auf dem Stack geschrieben wird (Abbildung 3).



3
2
1

Abbildung 3: Schritt drei

Da jetzt auch rechts nicht zu machen ist, kehrt der Algorithmus wieder zu der (immer noch lokalen) Wurzel (*) zurück. Da er bereits die beiden Äste (war ja auch nicht gerade viel) abgearbeitet hat, bleibt ihm nichts anderes übrig, als den aktuellen Knoten auf dem Stack zu speichern. Und zu seiner Wurzel zurückzukehren (Abbildung 4).



*
3
2
1

Abbildung 4: Schritt vier

Nun ist der Algorithmus wieder an seinem Ursprungspunkt angelangt, es bleibt also nur noch die „+“ Operation übrig, die auf dem Stack geschrieben wird (Abbildung 5).

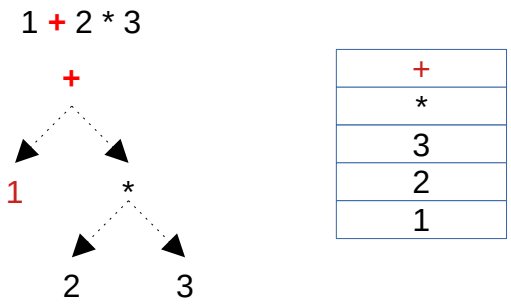


Abbildung 5: Schritt fünf

Wir setzen die entsprechenden vm Befehle manuell in einem Vector ein, und lassen die VM das Ergebnis berechnen (Abbildung 6). Beachte - da die Befehle in dem Vector in der Reihenfolge von oben nach unten abgearbeitet werden, ist deren Sortierung in dem Vector umgekehrt da ein folgendes Element das vorhergehende nach unten drückt.

```
17 void testVM() {
18
19     // 1+2*3
20   {
21     vector<Code> vc{
22       newPush(1),
23       newPush(2),
24       newPush(3),
25       newMult(),
26       newPlus(),
27     };
28        Optional<int> res = VM(vc).run();
29     printf("1 + 2 * 3 :");
30     showVMRes(res);
31   }
32 }
```

```
1 + 2 * 3 :
VM stack (top):7

...Program finished with exit code 1
Press ENTER to exit console.
```

Abbildung 6 Prüfung des Ergebnisses

2) Programmtechnische Umsetzung

Der Trick ist, hier analog zu der bereits bekannten Funktionalität der ***pretty(...)*** Methode vorzugehen und den AST-Baum automatisch abzuarbeiten.

Zuerst brauchen wir allerdings eine geeignete AST Struktur. Wir benutzen den Parser, der bereits aus der vorangegangenen Aufgabe bekannt ist um uns mit dessen Hilfe einem entsprechenden Baum zu erstellen. Ist eine geeignete Baumstruktur vorhanden, kann diese automatisch durch rekursive Verwendung der ***toVm()*** Methode abgearbeitet werden.

Zuerst rufen wir die Methode ***_toVm()*** auf des Wurzelelements unseres AST Baumes (Abbildung 7). Da wir hier objektorientiert arbeiten, weißt die Laufzeitumgebung das an dieser Stelle die Methode ***toVm()*** des hier verwendeten Objekts , also ***MultExp***, aufzurufen ist.

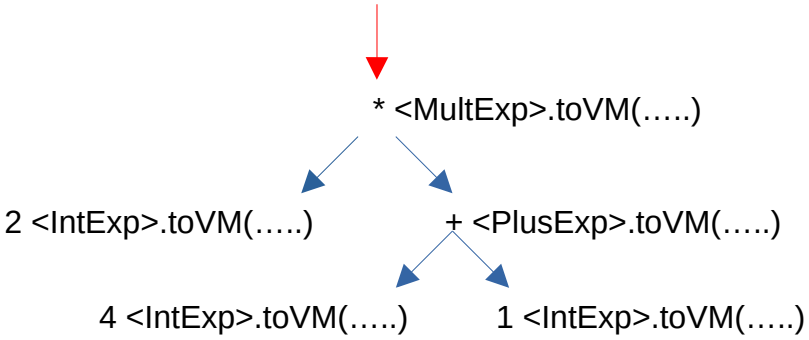


Abbildung 7:

In dem Wurzelknetton, da es kein Blatt ist, wird zuerst wider die ***toVm()*** Methode des linken Astes aufgerufen (Abbildung 8). Auch hier weist die Laufzeitumgebung welche Funktion für den entsprechenden Objekt zuständig ist.

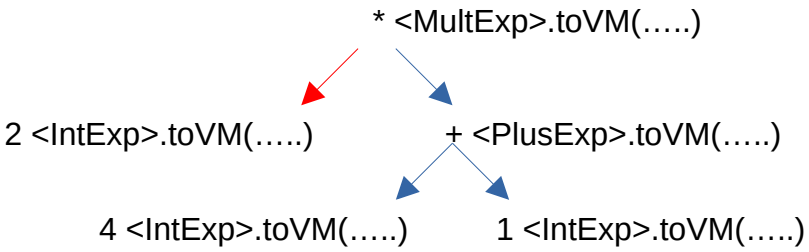


Abbildung 8:

Da der linke Ast nur ein Blatt enthält, gibt diese seine Daten an dem Aufrufer zurück und die Programmausführung kehrt zu diesem zurück. Jetzt ruft der Wurzelknoten wiederum die ***toVM()*** Methode des rechten Astes (Abbildung 9)

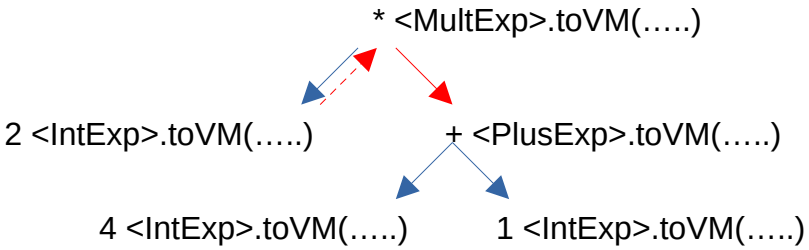


Abbildung 9:

Der da rechte Knoten kein Blatt ist, wird hier zuerst wiederum die ***toVm()*** Methode des linken Astes aufgerufen (Abbildung 10).

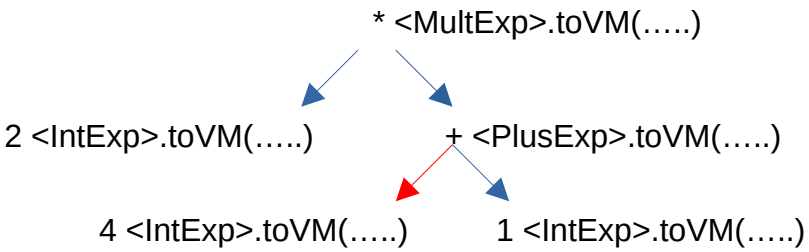


Abbildung 10:

Da dieser ein Blatt ist, übergibt er seine Daten an den Aufrufer und die Verarbeitung kehrt zu diesem zurück (Abbildung 11).

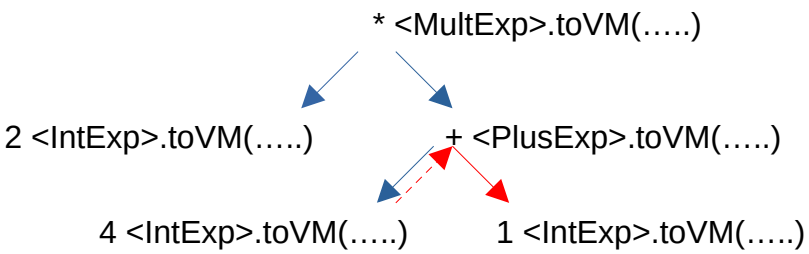


Abbildung 11:

Jetzt befindet sich die Anwendungsausführung wieder in einem Endknoten (Abbildung 12). so werden auch hier die Daten zurück an dem Aufrufer übergeben. Der Aufrufer (PlusExp) kann jetzt wiederum seine Daten (und die seiner Äste) an den Wurzelknoten übergeben so das auch schließlich dieser seine eigene Daten dem neu erstellten Datensatz zufügen kann.

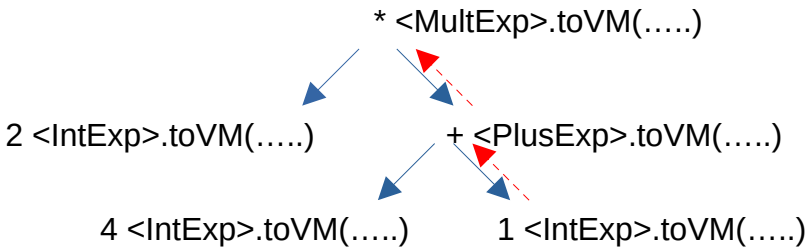


Abbildung 12:

Wie bringt man den abgearbeiteten AST Baum in die Virtuellemaschine? Hier gibt es zwei Möglichkeiten. Entweder werden die Befehle für die Virtuellemaschine direkt aus den Knoten/blättern in den Stack dieser geladen, oder die Daten, in diesem Fall Funktionsaufrufe, werden in einem Vector gesprichert und dieser letztendlich dann in einem VM Stack umgewandelt.