

Tutorium 14: Bytecode & Wiederholung

David Kaufmann

15. Februar 2023

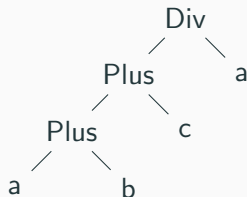
Tutorium Programmierparadigmen am KIT

Aufgabe

Wir bauen den Parser von letzter Woche weiter...

- Zu jeder Kategorie eine Klasse
- Jeder Blatttyp und innerer Knoten ein Konstruktor
- Unterklassen für Kategorie sind Alternativen
- Baum soll links-assoziativ sein

$$(a + b + c)/a$$



$$E \rightarrow T \ EList$$

$$EList \rightarrow \epsilon \mid + \ T \ EList \mid - \ T \ EList$$

$$T \rightarrow F \ TList$$

$$TList \rightarrow \epsilon \mid * \ F \ TList \mid / \ F \ TList$$

$$F \rightarrow \text{num} \mid (\ E \)$$

- Der Parser soll einen AST zurückgeben
- Verwendet die Klassen aus ./demos/compiler/Exp.java
- Der Baum soll wie auf letzter Folie aussehen

- PP beschäftigt sich (bis auf Typinferenz) nur kurz mit semantischer Analyse
- Typchecks/-inferenz, Namensanalyse
- \leadsto weiterführende (Master-)Vorlesungen am IPD

Java-Bytecode

Umgekehrte Polnische Notation

Schreibweise für Ausdrücke, bei der zuerst die Operanden und dann die auszuführende Operation angegeben wird.

Ist die natürliche Darstellung für Stackmaschinen

- $2 * 3$
- $2 * 3 + 4$
- $7 + 3 * (3 + 2)$

Umgekehrte Polnische Notation

Schreibweise für Ausdrücke, bei der zuerst die Operanden und dann die auszuführende Operation angegeben wird.

Ist die natürliche Darstellung für Stackmaschinen

- $2 * 3$
 - $2 * 3 + 4$
 - $7 + 3 * (3 + 2)$
- $2\ 3\ *$

Umgekehrte Polnische Notation

Schreibweise für Ausdrücke, bei der zuerst die Operanden und dann die auszuführende Operation angegeben wird.

Ist die natürliche Darstellung für Stackmaschinen

- $2 * 3$

- $2\ 3\ *$

- $2 * 3 + 4$

- $2\ 3\ * \ 4\ +$

- $7 + 3 * (3 + 2)$

Umgekehrte Polnische Notation

Schreibweise für Ausdrücke, bei der zuerst die Operanden und dann die auszuführende Operation angegeben wird.

Ist die natürliche Darstellung für Stackmaschinen

- $2 * 3$

- $2 * 3 + 4$

- $7 + 3 * (3 + 2)$

- $2\ 3\ *$

- $2\ 3\ * \ 4\ +$

- $7\ 3\ 3\ 2\ +\ *\ +$

- `X_const_i`: läd Konstante für $i \in [-1, 5]$ auf den Stack
- `bipush <i>`: läd Konstante für $i \in [-127, 128]$
- `Xload <x>`, `Xstore <x>`: lesen/schreiben von lokalen Variablen (für Integer gibt es eigenen Opcode für $x \in [0, 3]$)
- `Xmul`, `Xadd`, ...: Arithmetik

```
void calc(int x, int y) {  
    int z = x + 2;  
    y = z + x * 3;  
}
```

Variablen x, y, z in lokalen

Variablen 1, 2, 3 gespeichert

```
void calc(int x, int y) {  
    int z = x + 2;  
    y = z + x * 3;  
}
```

Variablen x, y, z in lokalen
Variablen 1, 2, 3 gespeichert

```
iload_1  
iconst_2  
iadd  
istore_3  
iload_1  
iconst_3  
imul  
iload_3  
iadd  
istore_2
```

- `goto label`: unbedingter Sprung
- `if_icmpOP label`: bedingter Sprung, der die ersten beiden Integer auf dem Stack vergleicht
- `ifOP label`: bedingter Sprung, der erstes Stackelement mit 0 vergleicht
- `ireturn`: gibt einen Integer zurück

$OP \in \{eq, ge, gt, le, lt\}$

```
public int fibs(int steps) {  
    int last0 = 1;  
    int last1 = 1;  
    while (--steps > 0) {  
        int t = last0 + last1;  
        last1 = last0;  
        last0 = t;  
    }  
    return last0;  
}
```

```
iconst_1  
istore_2  
iconst_1  
istore_3  
loop_begin:  
iinc 1 1  
iload_1  
ifle after_loop  
iload_2  
iload_3  
iadd  
istore_4  
iload_2  
istore_3 iload 4  
istore_2  
goto loop_begin  
after_loop:  
iload_2  
ireturn
```

Bei `&&` und `||` Kurzauswertung beachten!

Löst man über ein weiteres label, zu dem man nur springt, wenn noch nicht entschieden

Schreibt den Bytecode für:

```
if (a == 0 && b == a + 1)
```

Bei `&&` und `||` Kurzauswertung beachten!

Löst man über ein weiteres label, zu dem man nur springt, wenn noch nicht entschieden

Schreibt den Bytecode für:

```
if (a == 0 && b == a + 1)
```

Für Negation einfach Label vertauschen

- `this` kann mit `aload_0` geladen werden
- `invokevirtual #dest:` ruft in `dest` spezifizierte Methode auf
- `putfield name:type:` schreibt Wert von Typ `type` in Klassenvariable mit Name `name`. Auf dem Stack müssen der Wert und die Objectreferenz liegen

- `newarray type`: erzeugt ein neues Array. Verwendet das oberste Element auf dem Stack als Länge
- `astore`: speichern einer Referenz (z.B. Array)
- `iaload`: lädt Wert von Index aus Array. Stack muss von oben belegt sein mit: Index, Array Referenz.
- `iastore`: speichert Integer an Index in Array. Stack muss aussehen von oben: Wert, Index, Array Referenz

Klausuraufgabe SS17 Aufgabe 12

Evaluation

- Haskell: SS17 Nr. 1, 2
- Prolog: ÜB 7 Nr. 3
- β -Reduktion: SS17 Nr. 6a
- Lambda: SS18 Nr. 4
- Unifikation: SS17 Nr. 4
- Typinferenz: ÜB 9 Nr. 4
- MPI: SS17 Nr. 7
- Java: WS1718 Nr. 7
- Design By Contract: WS1718 Nr. 8
- Parser: SS20 Nr. 10
- Bytecode: WS2021 Nr. 8

Ende

- ÜB-Korrekturen: Blatt 13 korrigiere ich noch
- Klausur: 31.03.2023, 11:30
- Tutoriumsfolien, -code, etc.:
github.com/KaufmannDavid/propa-tut
- Fragen auch gerne an `david.kaufmann@student.kit.edu` :)

Danke fürs Kommen und eine erfolgreiche Prüfungsphase!