Einführung in den Compilerbau

Prof. Dr.-Ing. Andreas Koch
David Volz, Tim Noack, Jan Braun

TECHNISCHE UNIVERSITÄT DARMSTADT

Theorie
Wintersemester 23/24
Übungsblatt 1

Abgabe bis Sonntag, 12.11.2023, 18:00 Uhr (MEZ)

Auf diesem Aufgabenblatt sollen Sie sich mit der Matrix and Vector Language, kurz MAVL, vertraut machen. Studieren Sie bitte zunächst die MAVL-Sprachspezifikation, die Sie im Moodle-Kurs der Veranstaltung finden.

Aufgabe 1.1: MAVL-Syntax (19 Punkte)

Die MAVL-Sprachspezifikation enthält nur eine informelle Beschreibung der Syntaxelemente der Sprache. In den folgenden Teilaufgaben sollen Sie einige der Syntaxelemente in Produktionen einer kontextfreien Grammatik überführen.

Abweichend von den Vorlesungsfolien soll die Grammatik jedoch in *erweiterter* Backus-Naur-Form (EBNF) beschrieben werden: Sie können und sollen in den Produktionen den ?-Operator (0 oder 1 Wiederholung) und den *-Operator (0 oder mehr Wiederholungen) verwenden. Beispielsweise können Sie eine Sequenz eines Nichtterminals `A` statt durch

```
A ::= A single-A
   |  single-A
```

auch einfacher durch

```
A ::= single-A (single-A)*
```

ausdrücken.

Verwenden Sie bitte folgende Terminalsymbole (Tokens):

Terminalsymbole	Bedeutung
`ID`	Bezeichner, z.B. `someVar_42`
`INT`, `FLOAT`, `BOOL`, `STRING`	vorzeichenlose Literale des entsprechenden Typs, z.B. `17`, `3.14`, `false`, "`foo`"
`'function'`, `'val'`, `'var'`, `'for'`, `'if'`, `'else'`, `'return'`, `'foreach'`, `'switch'`, `'case'`, `'default'`, `'record'`	Schlüsselwörter
`'int'`, `'float'`, `'bool'`, `'void'`, `'string'`, `'matrix'`, `'vector'`	Schlüsselwörter für eingebaute Typen
`'('`, `')'`, `'{'`, `'}'`, `'['`, `']'`, `'<'`, `'>'`	Klammern
`','`, `':'`, `';'`, `'='`, `'~'`, `'#'`, `'.*'`, `'+'`, `'-'`, `'*'`, `'/'`, `'^'`, `'<='`, `'>='`, `'=='`, `'!='`, `'!'`, `'&'`, `'|'`, `'?'`, `'@'`	Symbole und Operatoren
`'.rows'`, `'.cols'`, `'.dimension'`	Spezielle Dimensionsoperatoren

Whitespace-Symbole und Kommentare brauchen Sie nicht zu beachten.
Wichtig: Lassen Sie in Ihrer Lösung nicht die einfachen Anführungsstriche weg – sie dienen zur Unterscheidung der Tokens von den Operatoren der EBNF.

Hinweis: Beachten Sie, dass Sie die in der Spezifikation angegebenen kontextuellen Einschränkungen, z.B. Typkompatibilität, hier nicht prüfen müssen. Verwenden Sie eine möglichst kompakte Beschreibung der Syntaxelemente. Die Grammatik darf durch Ihre Produktionen mehrdeutig werden. Ignorieren Sie für diese Aufgabe außerdem § 7.9 der MAVL-Spezifikation: Ihre Produktionen müssen nicht die Mehrfachanwendung von unären Operatoren ausschließen.

1.1a) `expr` (5 Punkte)

Das Nichtterminal `expr` dient zur Erkennung eines beliebig komplexen Ausdrucks wie z.B. `a * b + (c - m[3][1])`. In einer kontextfreien Grammatik lassen sich Ausdrücke als rekursive Produktionen beschreiben: Ein Ausdruck ist entweder ein atomarer Ausdruck oder entsteht durch die Anwendung eines Operators auf weitere (atomare oder zusammengesetzte) Ausdrücke.
Hinweis: Die Operatorpräzedenzen müssen Sie hier nicht beachten.

`expr` wird durch folgende Produktion beschrieben:

```
expr ::= ID | INT | BOOL | FLOAT | STRING
       |  '(' expr ')'
       | ...
       |  addExpr
       |  ternaryExpr
       |  subvectorExpr
       | ...
```

Geben Sie Produktionen für die Nichtterminale `addExpr` (Addition), `ternaryExpr` (Ternärer Operator) und `subvectorExpr` (Subvektor-Operator) an.

1.1b) `type` (4 Punkte)

Das Nichtterminal `type` dient zur Erkennung von Typbezeichnern wie z.B. `vector<int>[3]`. `type` wird durch folgende Produktion beschrieben:

```
type ::= primitiveType
       |  vectorType
       | ...
```

Geben Sie Produktionen für die Nichtterminale `primitiveType` (primitive Typen) und `vectorType` (Vektortypen) an.

1.1c) `stmt` (6 Punkte)

Das Nichtterminal `stmt` dient zur Erkennung von Befehlen und wird durch folgende Produktion beschrieben:

```
stmt ::= varAssign
       |  compoundStmt
       |  foreachStmt
       | ...
```

Geben Sie Produktionen für die Nichtterminale `varAssign` (Variablenzuweisung), `compoundStmt` (Befehlsblock), sowie `foreachStmt` (Foreach-Scheife) an.

1.1d) Konstante Ausdrücke (4 Punkte)

Konstante Ausdrücke (vgl. MAVL-Spezifikation Abschnitt 7.3) sind aus Sicht des Parsers sonst ganz normale Ausdrücke, die Überprüfung auf Konstantheit findet im MAVL-Compiler erst während der Kontextanalyse statt. Für diese Teilaufgabe sollen Sie jedoch eine Produktion `constExpr` angeben, die bereits in der syntaktischen Analyse nur konstante Integer-Ausdrücke akzeptiert.

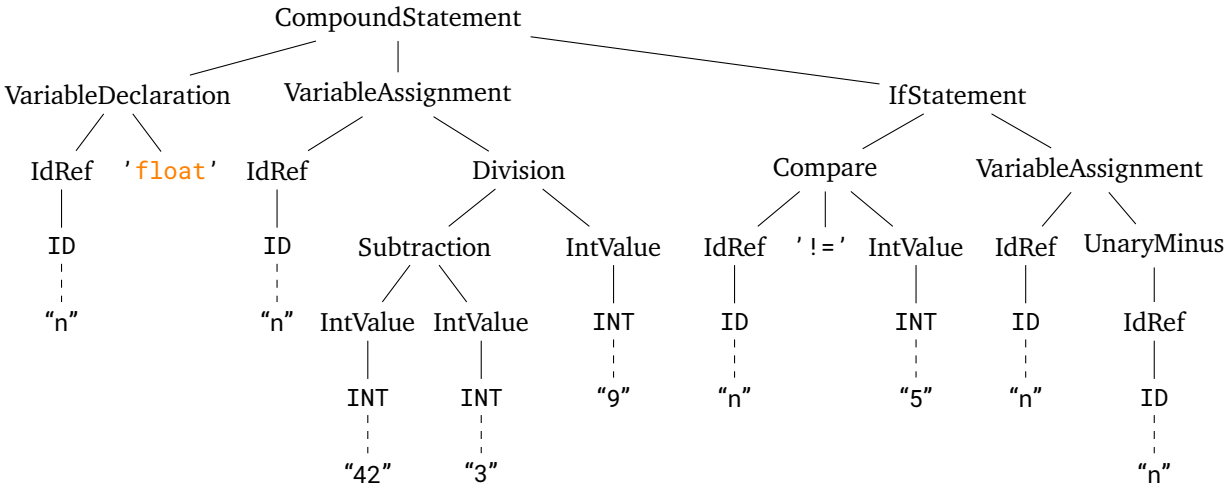
Aufgabe 1.2: AST → MAVL (10 Punkte)

Abstrakte Syntaxbäume (engl. Abstract Syntax Trees (AST)) sind eine weitverbreite Zwischendarstellung, die nur essentielle Informationen enthält und Details der konkreten Syntax einer Programmiersprache abstrahiert.

In dieser Aufgabe zeigen wir Ihnen eine mögliche Repräsentation von MAVL-Code als AST. Die darin verwendeten AST-Knoten korrespondieren auf natürliche Weise mit den in der Spezifikation beschriebenen Syntaxelementen.

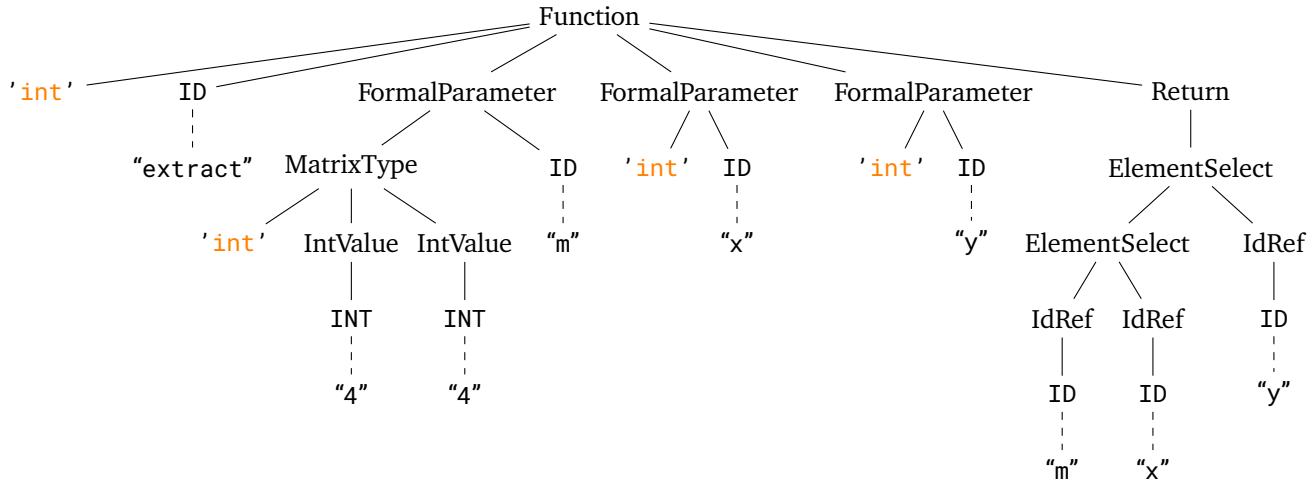
1.2a) AST → MAVL (5 Punkte)

Geben Sie den zum folgenden AST zugehörigen MAVL-Code an.



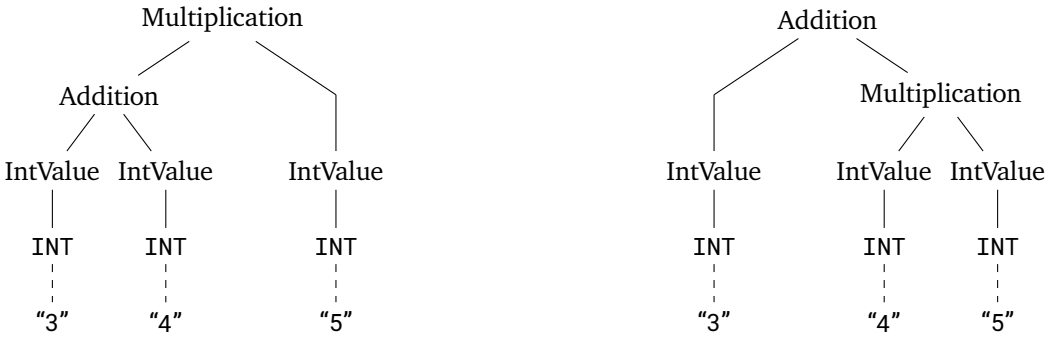
1.2b) AST → MAVL (5 Punkte)

Geben Sie den zum folgenden AST zugehörigen MAVL-Code an.



Aufgabe 1.3: Ausdrücke (11 Punkte)

Ausdrücke in typischen Programmiersprachen lassen sich einfach durch mehrdeutige Grammatiken beschreiben, die aber als Grundlage für die syntaktische Analyse ungeeignet sind. Beispielsweise existieren damit für den Ausdruck `3 + 4 * 5` zwei mögliche (abstrakte) Syntaxbäume:



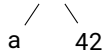
Die Auswertung des Ausdrucks gemäß des linken ASTs verletzt die "Punkt-vor-Strich"-Regel und liefert den Wert 35. Der rechte AST repräsentiert die erwartete Auswertung des Ausdrucks zum Wert 23.

Die MAVL-Spezifikation stellt eine eindeutige Auswertungsreihenfolge (und folglich: eine eindeutige Struktur des ASTs) durch die Definition von Operatorpräzedenzen sicher. Innerhalb einer Präzedenzebene werden Ausdrücke meistens von links nach rechts ausgewertet (Ausnahme: Der `^`-Operator ist rechtsassoziativ.)

Hinweis: Verwenden Sie in dieser Aufgabe folgende AST-Knoten:

> **And, Or, Not, Addition, Subtraction, Multiplication, Division, Exponentation, Compare, Ternary, Transpose, Rows, Cols, DotProduct, MatrixMult, StructureInit, ElementSelect, SubVector, SubMatrix.**

Hängen Sie die Bezeichner und Integer-Literale ohne die Knoten IdRef, IntValue, `ID` und `INT` in Ihrer Lösung direkt an die entsprechenden Operatorknoten, wie z.B.:



1.3a) MAVL → AST (5 Punkte)

Zeichnen Sie den AST für den MAVL-Ausdruck

```
!s & (a >= b - c / d | t ? false : true)
```

1.3b) MAVL → AST (4 Punkte)

Zeichnen Sie den AST für den MAVL-Ausdruck

```
vec .* (~mat)[3]{-1:1:1}
```

1.3c) MAVL-Rechnung (2 Punkte)

Gegeben seien folgende Wertdefinitionen:

```
val float a = 12.34; val float b = 13; val float c = 37; val float d = 42;
val bool s = false; val bool t = false;
val matrix<int>[4][4] mat = [[1, 0, 0, 2],
                             [0, 1, 0, 3],
                             [0, 0, 1, 8],
                             [4, 5, 6, 1]];
val vector<int>[3] vec = [9, 8, 7];
```

Welche Werte liefern die Ausdrücke aus den Teilaufgaben a) und b)?