Lehrstuhl für Softwaretechnik und Programmiersprachen
Professor Dr. Michael Leuschel

Compilerbau - Wintersemester 2020/21

Übungsblatt 8 - Musterlösung

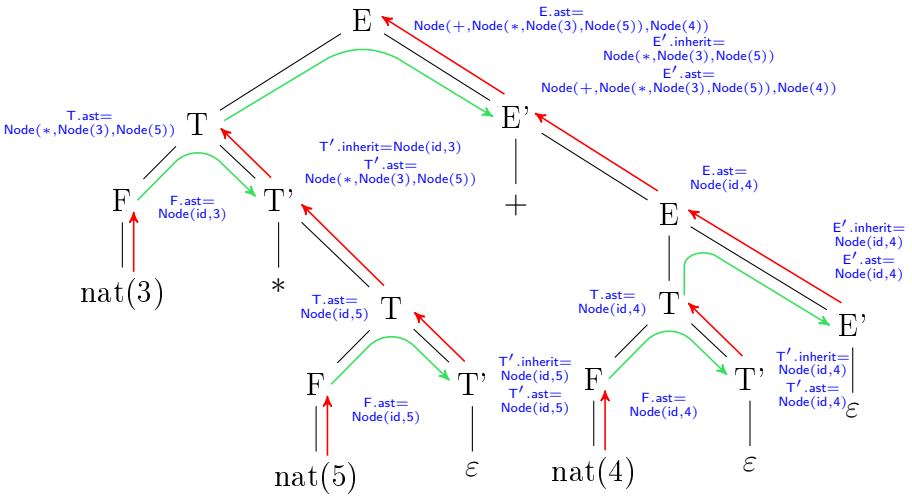
Aufgabe 8.1

a) Erweitern Sie die Grammatik G' um semantische Aktionen, um einen AST zu erzeugen, welcher G entspricht.

1)	E → TE'	{ E.ast = E'.ast, E'.inherit = T.ast}
2)	E' → +E	{ E'.ast = new Node('+', E'.inherit, E.ast) }
3)	E' → ε	{ E'.ast = E'.inherit }
4)	T → FT'	{ T.ast = T'.ast, T'.inherit = F.ast}
5)	T' → ∗T	{ T'.ast = new Node('*', T'.inherit, T.ast) }
6)	T' → ε	{ T'.ast = T'.inherit }
7)	F → nat	{ F.ast = new Node('nat', nat.lexValue) }
8)	F → id	{ F.ast = new Node('id', id.lexValue) }
9)	F → (E)	{ F.ast = E.ast }

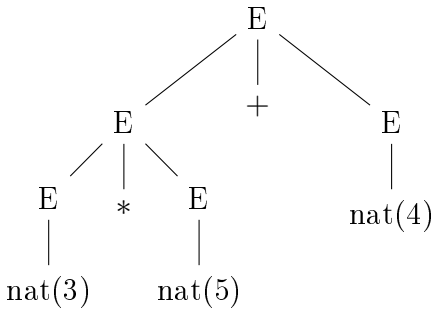
b) Geben Sie einen Parsebaum für die Eingabe $3 * 5 + 4$ für G sowie für G' an.

Syntaxbaum für G':

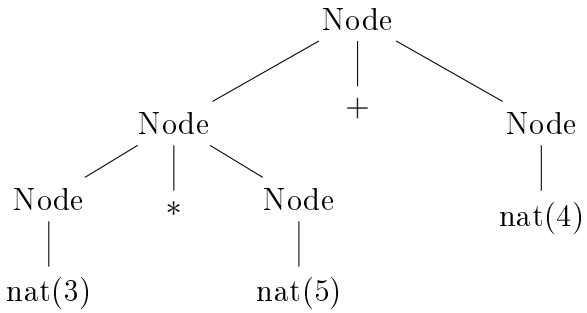


Hinweis: Attribute stehen nur der Übersichtlichkeit an einigen Stellen Links.

Syntaxbaum für G:



Syntaxbaum durch semantische Aktionen:



Node(+,Node(*,Node(3),Node(5)),Node(4))

[1]

c) Der Abhängigkeitsgraph wurde in Form von farbigen Pfeilen in den Syntaxbaum für G' eingezeichnet
Synthetisierte Attribute (von unten nach oben) sind dort in rot markiert, vererbte Attribute (von oben nach unten oder links nach rechts) dagegen in grün. Dabei gibt es keine Zyklen. Es handelt sich hier um eine L-Attributgrammatik, aber um keine S-Attributgrammatik, da beispielsweise mit E'.inherit = T.ast in (a1) eine Vererbung stattfindet.

Aufgabe 8.2

a) Geben Sie eine konkrete Grammatik G' mit korrekter Operatorpräzedenz an, mit welcher sich alle Wörter in G erzeugen lassen.

E → E==C | C
C → C<T | C>T | T
T → T + F | T - F | F
F → F * L | F / L | L
L → true | false | num

b) Erweitern Sie die Grammatik G' um semantische Aktionen um einen AST zu erzeugen (welcher G entspricht)

[1] Danke an Florian Dittrich für den Latex-Tipp mit dem Baum

1)	E → E==C	{ E.ast = new Node(==, E.ast, C.ast)}
2)	E → C	{ E.ast = C.ast}
3)	C → C<T	{ C.ast = new Node(<, C.ast, T.ast)}
4)	C → C>T	{ C.ast = new Node(>, C.ast, T.ast)}
5)	C → T	{ C.ast = T.ast}
6)	T → T + F	{ T.ast = new Node(+, T.ast, F.ast)}
7)	T → T - F	{ T.ast = new Node(-, T.ast, F.ast)}
8)	T → F	{ T.ast = F.ast}
9)	F → F * L	{ F.ast = new Node(*, F.ast, L.ast)}
10)	F → F / L	{ F.ast = new Node(/, F.ast, L.ast)}
11)	F → L	{ F.ast = L.ast}
12)	L → true	{ L.ast = new Node(true, true.LexValue)}
13)	L → false	{ L.ast = new Node(false, false.LexValue)}
14)	L → num	{ L.ast = new Node(num, num.LexValue)}

c) Erweitern Sie die Grammatik G um semantische Aktionen für einen AST Interpreter.

1)	$E_0 \rightarrow E_1 + E_2$	{ $E_0.value = E_1.value + E_2.value$}
2)	$E_0 \rightarrow E_1 - E_2$	{ $E_0.value = E_1.value - E_2.value$}
3)	$E_0 \rightarrow E_1 * E_2$	{ $E_0.value = E_1.value * E_2.value$}
4)	$E_0 \rightarrow E_1/E_2$	{ $E_0.value = E_1.value/E_2.value$}
5)	$E_0 \rightarrow E_1 < E_2$	{ $E_0.value = E_1.value < E_2.value$}
6)	$E_0 \rightarrow E_1 > E_2$	{ $E_0.value = E_1.value > E_2.value$}
7)	$E_0 \rightarrow E_1 == E_2$	{ $E_0.value = E_1.value == E_2.value$}
8)	$E_0 \rightarrow true$	{ $E_0.value = true$}
9)	$E_0 \rightarrow false$	{ $E_0.value = false$}
10)	$E_0 \rightarrow num$	{ $E_0.value = num.LexValue$}

d) Erweitern Sie die Grammatik G um semantische Aktionen für einen Typechecker.

1)	$E_0 \rightarrow E_1 + E_2$	{ $E_0.type = int; check(E_1.type, int); check(E_2.type, int);$}
2)	$E_0 \rightarrow E_1 - E_2$	{ $E_0.type = int; check(E_1.type, int); check(E_2.type, int);$}
3)	$E_0 \rightarrow E_1 * E_2$	{ $E_0.type = int; check(E_1.type, int); check(E_2.type, int);$}
4)	$E_0 \rightarrow E_1/E_2$	{ $E_0.type = int; check(E_1.type, int); check(E_2.type, int);$}
5)	$E_0 \rightarrow E_1 < E_2$	{ $E_0.type = boolean; check(E_1.type, int); check(E_2.type, int);$}
6)	$E_0 \rightarrow E_1 > E_2$	{ $E_0.type = boolean; check(E_1.type, int); check(E_2.type, int);$}
7)	$E_0 \rightarrow E_1 == E_2$	{ $E_0.type = boolean; check(E_1.type, E_2.type);$}
8)	$E_0 \rightarrow true$	{ $E_0.type = boolean;$}
9)	$E_0 \rightarrow false$	{ $E_0.type = boolean;$}
10)	$E_0 \rightarrow num$	{ $E_0.type = int;$}

Mit:

```
check(Type actual, Type expected) {
    if (actual!=expected) {
```

```
            throw new TypeException(actual, expected);
        }
    }
```

Aufgabe 8.3

a) Geben Sie die Symboltabellen für das Programm (also global), die Funktion f und Funktion neg an.

Global

Symbol	Typ
x	int
z	bool
neg	$bool \Rightarrow bool$
f	$int \Rightarrow int$

f

Symbol	Typ
x	int

neg

Symbol	Typ
b	bool

b) Wie funktioniert der lookup in Zeile 6 ($z := neg(x + 5 < 4)$;) ? D.h. welche Tabellen werden mit welchen Schlüsseln in welcher Reihenfolge abgefragt ?
lookup(f,x)= int.
lookup(f,neg)= fail, lookup(neg, gloabal)= $bool \Rightarrow bool$ (x=5<4 ist vom typ bool).
lookup(f,z)= fail; lookup(f,z)=bool