

## Vorlagen für ahnen-Compiler

Der jeweils vorgegebene (immer gleiche) Parser soll (mit Attributen, Typen und zusätzlichen Prädikaten) zu verschiedenen Compilern erweitert werden, die Ahnen-Bezeichnungen (mutter, vater, grossmutter, grossvater, urgrossmutter, ... etc.) übersetzen (in bestimmte Zahlen, in eine Zwischendarstellung bzw. ins Englische).

In die noch leer gelassenen runden Klammern ( ) müssen geeignete Attribute eingetragen werden, vorzugsweise mit einem Bleistift (damit das Endergebnis auch nach ein paar kleinen Korrekturen mit einem Radiergummi noch lesbar ist).

### ahnen\_v2.g: Übersetzt in natürliche Zahlen (0, 1, 2, ...):

```

1 'nonterm' ahne (->INT )    -- Pfeil -> muss sein!
2   'rule' ahne (      ): ahne1(      )
3   'rule' ahne (      ): ahne2(      )
4   'rule' ahne (      ): ahne3(      )
5
6 'nonterm' ahne1(      )
7   'rule' ahne1(      ): "mutter"
8   'rule' ahne1(      ): "vater"
9
10 'nonterm' ahne2(      )
11  'rule' ahne2(      ): "gross" ahne1(      )
12
13 'nonterm' ahne3(      )
14  'rule' ahne3(      ): "ur" ahne2(      )
15  'rule' ahne3(      ): "ur" ahne3(      )
16
17 'root'
18  ahne (      ->ERG)      -- Pfeil -> muss sein!
19  print(ERG->      )      -- Pfeil -> kann sein.
```

### Erläuterungen:

nonterm-Prädikate darf man nur mit *Ausgabeparametern* („mit Parametern rechts vom Pfeil ->“) ausrüsten. Andere Prädikate darf man mit *Eingabeparametern* (links vom Pfeil) und *Ausgabeparametern* ausrüsten.

In Zeile 1 wird festgelegt, dass das nonterm-Prädikat ahne 0 Eingabeparameter und einen Ausgabe-parameter vom Typ INT hat.

In Zeile 18 wird das Prädikat ahne (entsprechend seiner Definition in Zeile 1) mit 0 Eingabeparametern und einem Ausgabeparameter ERG aufgerufen. Nachdem dieser Aufruf ausgeführt wurde, enthält die Variable ERG die Zahl, in die das Quellprogramm übersetzt wurde.

In Zeile 19 wird das (vordefinierte) Prädikat print aufgerufen. Es hat einen Eingabeparameter und 0 Ausgabeparameter. Es gibt den Wert seines Eingabeparameters zum Bildschirm aus.

**ahnen\_v3.g: Übersetzt in eine Zwischendarstellung des Typs ZAHNE**

```
20 'type' ZAHNE
21   mu
22   va
23   ur(ZAHNE)
24
25 'nonterm' ahne (      )
26   'rule' ahne (      ): ahne1(      )
27   'rule' ahne (      ): ahne2(      )
28   'rule' ahne (      ): ahne3(      )
29
30 'nonterm' ahne1(      )
31   'rule' ahne1(      ): "mutter"
32   'rule' ahne1(      ): "vater"
33
34 'nonterm' ahne2(      )
35   'rule' ahne2(      ): "gross" ahne1(      )
36
37 'nonterm' ahne3(      )
38   'rule' ahne3(      ): "ur" ahne2(      )
39   'rule' ahne3(      ): "ur" ahne3(      )
40
41 'root'
42   ahne (      ->ERG)
43   print(ERG->      )
```

Geben Sie ein paar Terme des Typs ZAHNE an (mindestens 5):

**ahnen\_v4.g: Übersetzt direkt in englische Strings**

```
44 'nonterm' ahne (          )
45   'rule' ahne (          ): ahne1(          )
46   'rule' ahne (          ): ahne2(          )
47   'rule' ahne (          ): ahne3(          )
48
49 'nonterm' ahne1(          )
50   'rule' ahne1(          ): "mutter"
51   'rule' ahne1(          ): "vater"
52
53 'nonterm' ahne2(          )
54   'rule' ahne2(          ): "gross" ahne1(          )
55
56
57
58 'nonterm' ahne3(          )
59   'rule' ahne3(          ): "ur" ahne2(          )
60
61
62   'rule' ahne3(          ): "ur" ahne3(          )
63
64
65
66 'action' conc2(Arg1: STRING, Arg2: STRING -> Erg: STRING)
67 -- Erg ist die Konkatenation von Arg1 und Arg2
68 -- conc2 wurde in C programmiert (Datei str_hand.c)
69
70 'root'
71   ahne (    ->ERG)
72   print(ERG->    )
```

**ahnen\_v5.g: ahnen\_v4.g: Übersetzt über Zwischendarstellung in englische Strings**

```
73 'type' ZAHNE
74     mu
75     va
76     ur(ZAHNE)
77
78 'nonterm' ahne (
79     'rule' ahne ( ): ahne1(
80     'rule' ahne ( ): ahne2(
81     'rule' ahne ( ): ahne3(
82
83 'nonterm' ahne1(
84     'rule' ahne1( ): "mutter"
85     'rule' ahne1( ): "vater"
86
87 'nonterm' ahne2(
88     'rule' ahne2( ): "gross" ahne1(
89
90 'nonterm' ahne3(
91     'rule' ahne3( ): "ur" ahne2(
92     'rule' ahne3( ): "ur" ahne3(
93
94 'action' aus(ZAHNE->) -- Pfeil -> muss nicht sein
95     'rule' aus( ):
96     'rule' aus( ):
97     'rule' aus( ):
98     'rule' aus( ):
99
100 -- Zwei in C programmierte Aktionen (text_io.c):
101 'action' Puts(String->)-- Gibt seinen String-Parameter aus
102 'action' Nl -- Gibt einen Zeilenwechsel aus
103             -- (Enn Ell, wie NewLine)
104
105 'root'
106     ahne( ->ERG)
107     aus (ERG-> )
```