Vorlagen für ancestor-Compiler

Der jeweils vorgegebene (immer gleiche) Parser soll (mit Attributen, Typen und zusätzlichen Prädikaten) zu verschiedenen Compilern erweitert werden, die ancestor-Bezeichnungen (`mother`, `father`, `grandmother`, `grandfather`, `greatgrandmother`, ... etc.) übersetzen (in bestimmte Zahlen, in eine Zwischendarstellung bzw. ins Deutsche).

In die noch leer gelassenen runden Klammern `(  )` müssen geeignete Attribute eingetragen werden, vorzugsweise mit einem **Bleistift**.

ancestor02: Übersetzt in natürliche Zahlen (0, 1, 2, ...):

```
 1 phrase ancestor (->int )
 2    rule ancestor (      ): ancestor1(      )
 3    rule ancestor (      ): ancestor2(      )
 4    rule ancestor (      ): ancestor3(      )
 5
 6 phrase ancestor1(      )
 7    rule ancestor1(      ): "mother"
 8    rule ancestor1(      ): "father"
 9
10 phrase ancestor2(      )
11    rule ancestor2(      ): "grand" ancestor1(      )
12
13 phrase ancestor3(      )
14    rule ancestor3(      ): "great" ancestor2(      )
15    rule ancestor3(      ): "great" ancestor3(      )
16
17 root
18    ancestor(->DISTANCE)
19    print DISTANCE
```

Erläuterungen:
`phrase`-Prädikate darf man nur mit *einem out-Parameter* (rechts vom Pfeil `->`) ausrüsten.
Andere Prädikate darf man mit *in-Parametern* (links vom Pfeil) und *out-Parameter n* ausrüsten.

In Zeile 1 wird festgelegt, dass das `phrase`-Prädikat `ancestor`
null in-Parameter und einen out-Parameter vom Typ `int` hat.

In Zeile 18 wird das Prädikat `ancestor`
mit null in-Parametern und einem out-Parameter `DISTANCE` aufgerufen.
Nach diesem Aufruf enthält die Variable `DISTANCE` die Zahl, in die die Quelldatei übersetzt wurde.

In Zeile 3 wird das Prädikat `print` aufgerufen.
Dieses Prädikat ist vordefiniert, sollte nur für *Test-Ausgaben* verwendet werden, wird ohne Klammern notiert, hat einen Parameter eines *beliebigen Typs* und gibt den zur Standardausgabe aus.

In den Regeln eines `phrase`-Prädikats bezeichnen string-Literale (wie z.B. `"mother"` oder `"great"`) *Eingaben*, die vom Parser eingelesen werden sollen.

ancestor03: Übersetzt in eine Zwischendarstellung des Typs AS_ancestor03

```
 1 type AS_ancestor03
 2    mo()
 3    fa()
 4    g(AS_ancestor03)
 5
 6 phrase ancestor (         )
 7    rule ancestor (         ): ancestor1(      )
 8    rule ancestor (         ): ancestor2(      )
 9    rule ancestor (         ): ancestor3(      )
10
11 phrase ancestor1(         )
12    rule ancestor1(         ): "mother"
13    rule ancestor1(         ): "father"
14
15 phrase ancestor2(         )
16    rule ancestor2(         ): "grand" ancestor1(      )
17
18 phrase ancestor3(         )
19    rule ancestor3(         ): "great" ancestor2(      )
20    rule ancestor3(         ): "great" ancestor3(      )
21
22 root
23    ancestor (-> AS)
24    print AS
```

Geben Sie ein paar Terme des Typs `AS_ancestor03` an (mindestens 5):

ancestor04: Übersetzt direkt in Listen von englischen Strings

```
 1 phrase ancestor (              )
 2    rule ancestor (         ): ancestor1(         )
 3    rule ancestor (         ): ancestor2(         )
 4    rule ancestor (         ): ancestor3(         )
 5
 6 phrase ancestor1(              )
 7    rule ancestor1(                 ): "mother"
 8    rule ancestor1(                 ): "father"
 9
10 phrase ancestor2(              )
11    rule ancestor2(                 ):
12       "grand" ancestor1(       )
13
14 phrase ancestor3(              )
15    rule ancestor3(         ):
16       "great" ancestor2(       )
17    rule ancestor3(         ):
18       "great" ancestor3(       )
19
20 proc out(L:string[])
21    // Outputs L
22    rule out(string[]):
23       "\n"
24    rule out(string[H::T]):
25       $H out(T)
26
27 root
28    ancestor(->LOS) // List Of Strings
29    out(LOS)
```

Erläuterungen:

Ein proc-Prädikat (a procedure) kann beliebig viele in-Parameter und beliebig viele out-Parameter haben. Das proc-Prädikat out hat einen in-Parameter vom Typ `string[]` und 0 out-Parameter.

In den Regeln eines proc-Prädikats werden string-*Literale* wie z.B. `"\n"` automatisch *ausgegeben*. Um den Wert einer `int`- oder `string`-*Variablen* wie z.B. `H` ausgeben zu lassen, muss man ein Dollarzeichen davor schreiben, z.B. `$H`.

Achtung: In Zeile 20 bezeichnet `string[]` den Typ *Liste von string-Werten*.
In Zeile 22 bezeichnet `string[]` dagegen eine *leere Liste* von string-Werten.

Das *Muster* `string[H::T]` (in Zeile 28) passt auf alle Listen von string-Werten, die aus einem head H (vom Typ `string`) und einem tail T (vom Typ `string[]`) bestehen.
Mit anderen Worten: Das Muster passt auf alle string-Listen, die mindestens *ein* Element enthalten.

Die Variablen-Namen `H` und T sind üblich, aber frei wählbar. Statt `string[H::T]` kann man z.B. auch `string[ErstesElement::RestDerListe]` oder `string[x::y]` schreiben.

ancestor05: Übersetzt in eine Zwischendarstellung und die ins Deutsche

```
 1 type AS_ancestor03
 2    mo()
 3    fa()
 4    g(AS_ancestor03)
 5
 6 root
 7    ancestor(->AS)
 8    trout(AS)
 9
10 phrase ancestor (              )
11    rule ancestor (      ): ancestor1(          )
12    rule ancestor (      ): ancestor2(          )
13    rule ancestor (      ): ancestor3(          )
14
15 phrase ancestor1(              )
16    rule ancestor1(      ): "mother"
17    rule ancestor1(      ): "father"
18
19 phrase ancestor2(               )
20    rule ancestor2(      ): "grand" ancestor1(        )
21
22 phrase ancestor3(               )
23    rule ancestor3(      ): "great" ancestor2(        )
24    rule ancestor3(      ): "great" ancestor3(        )
25
26
27 // Translation of abstract syntax into German and output
28 // (add 4 rules to complete)
29 proc  trout(AS_ancestor03)
30    rule trout(         ):
...
...
```