Zwei Co-Transformationen von Grammatiken und dazugehörigen Transformationsregeln

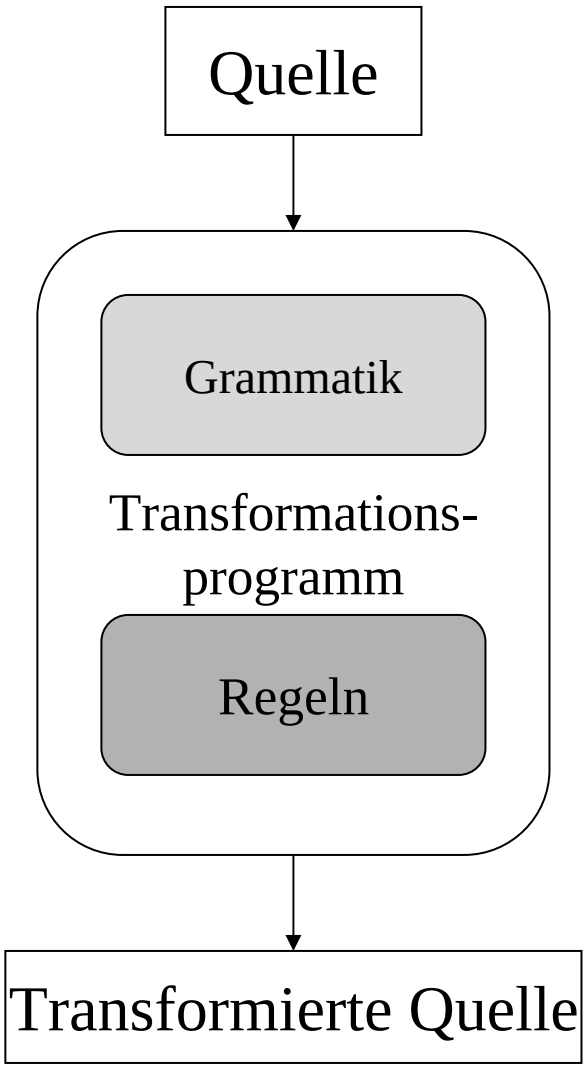
Wolfgang Lohmann

Universität Rostock

Überblick

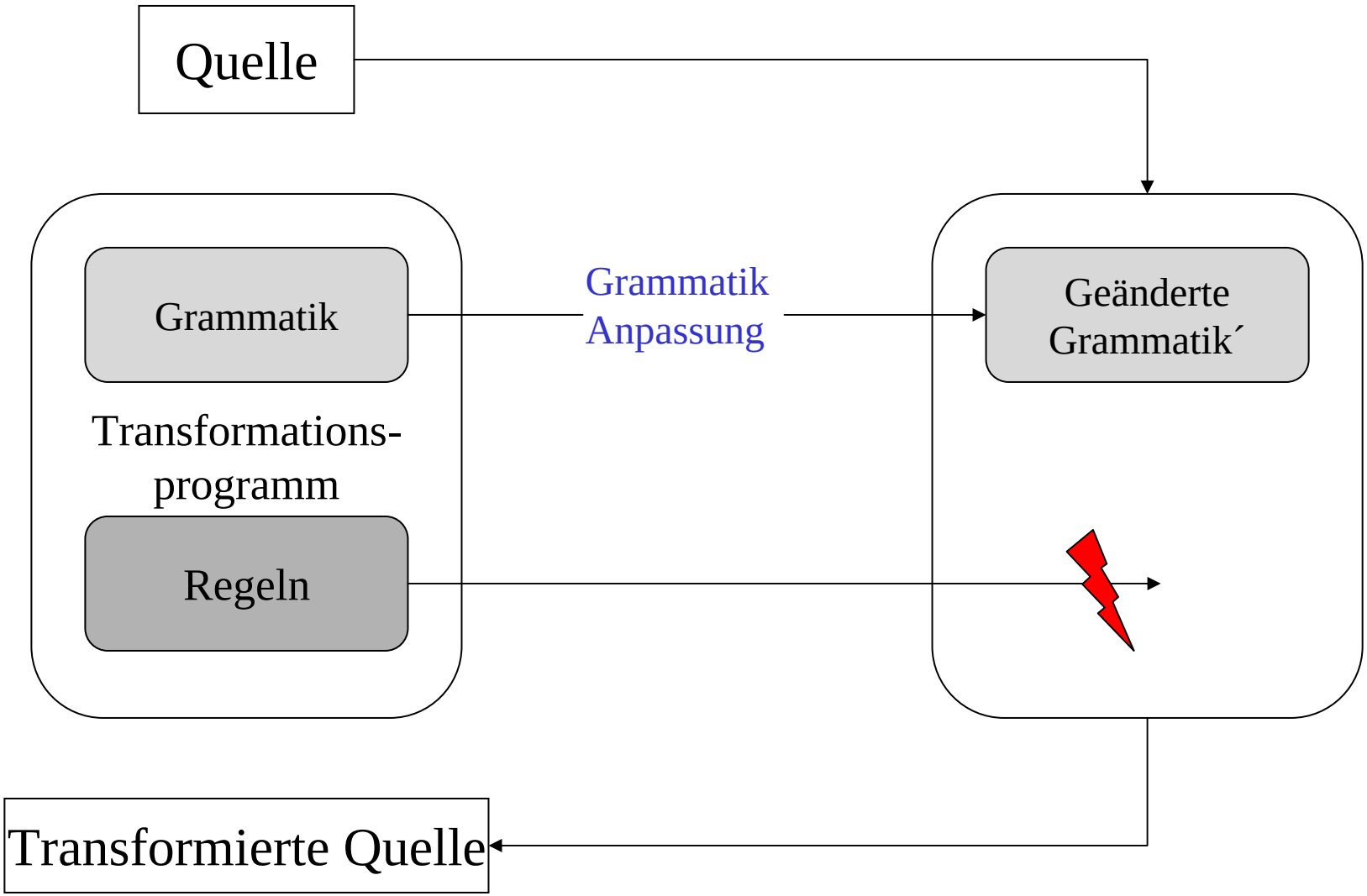
- Ausgangspunkt, Problem und Fragestellung
- Bezeichnung Co-Transformation
- Beispiel 1: Entfernung Linksrekursion
- Beispiel 2: Nachträgliches Einführen von Layouterhaltung in Transformationen
- Abschließende Bemerkung

Ausgangspunkt

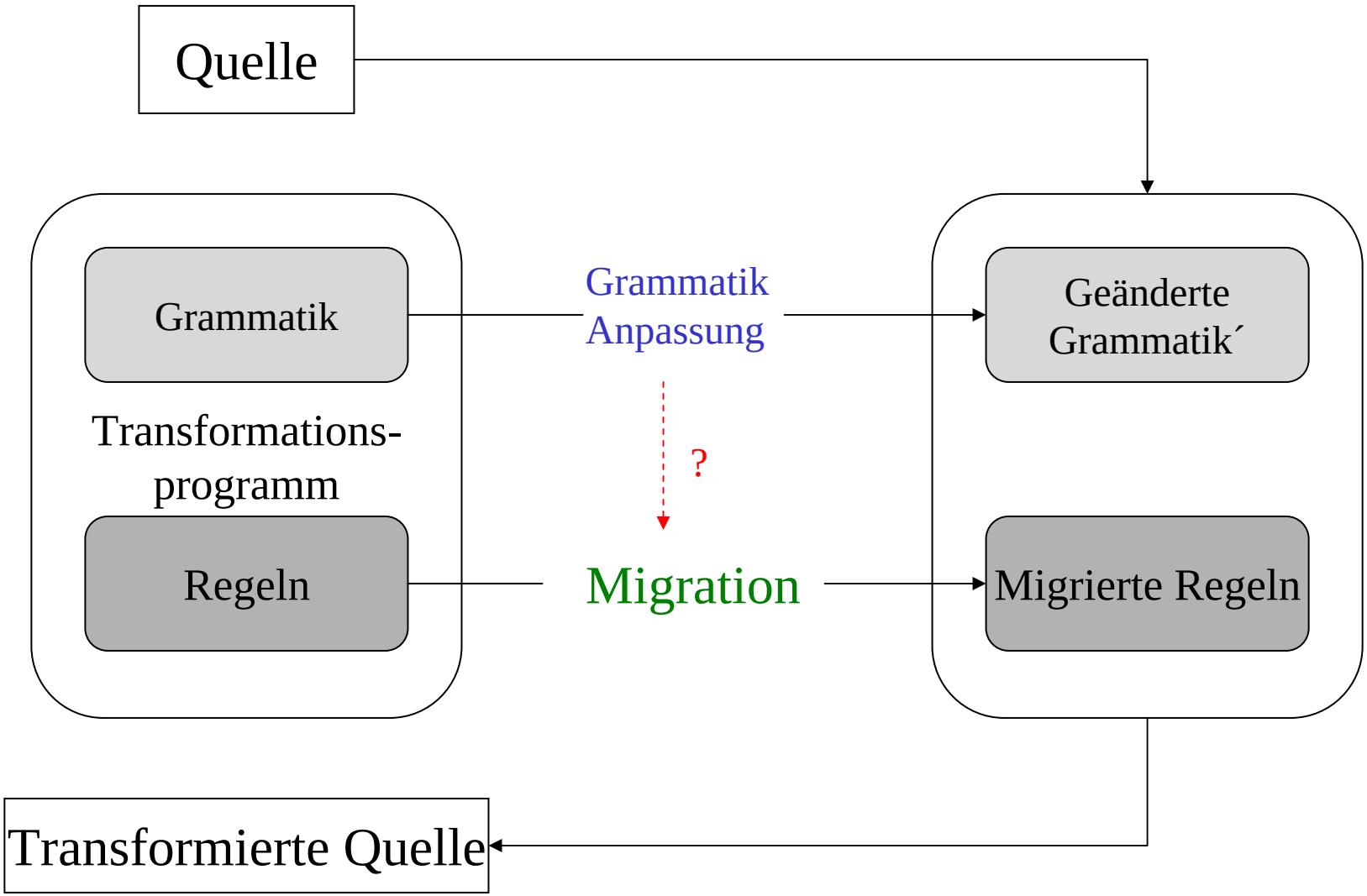


- Einfache kleine Analysewerkzeuge, Transformationen
  - Verwendung von Prolog (Pattern Matching, Top-Down)
  - Anpassung von vielen ähnlichen Programmen entsprechend einer Änderung in der Bibliothek
- Grammatiken als Softwareartefakte
  - Änderung,Weiterentwicklung, Debugging
  - Erstellung spezieller für Transformationen geeigneter Grammatiken (z.B. Agiles Parsing)
  - Lösungung von Problemen, zB. Linksrekursivität und Layouterhaltung

Problem

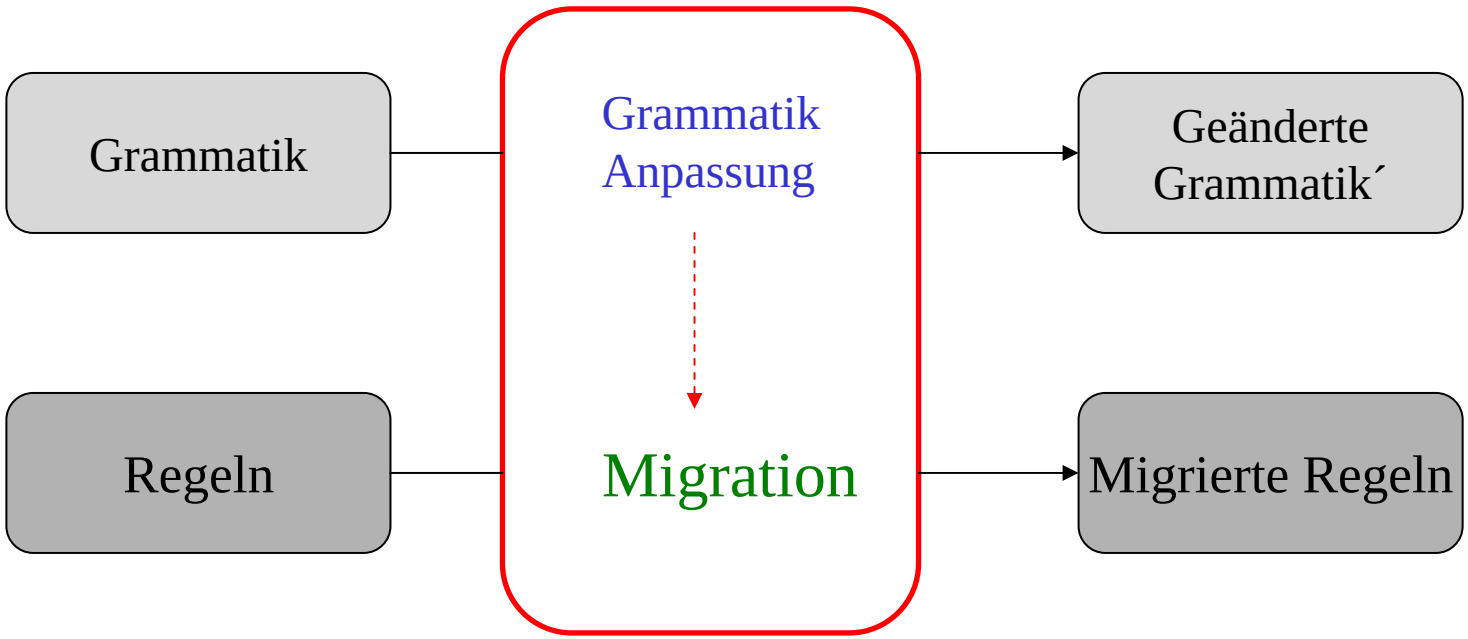

Quelle
Grammatik
Transformations-
programm
Regeln
Grammatik
Anpassung
Geänderte
Grammatik´
Transformierte Quelle

Frage


Quelle
Grammatik
Transformations-
programm
Regeln
Grammatik
Anpassung
?
Migration
Geänderte
Grammatik´
Migrierte Regeln
Transformierte Quelle

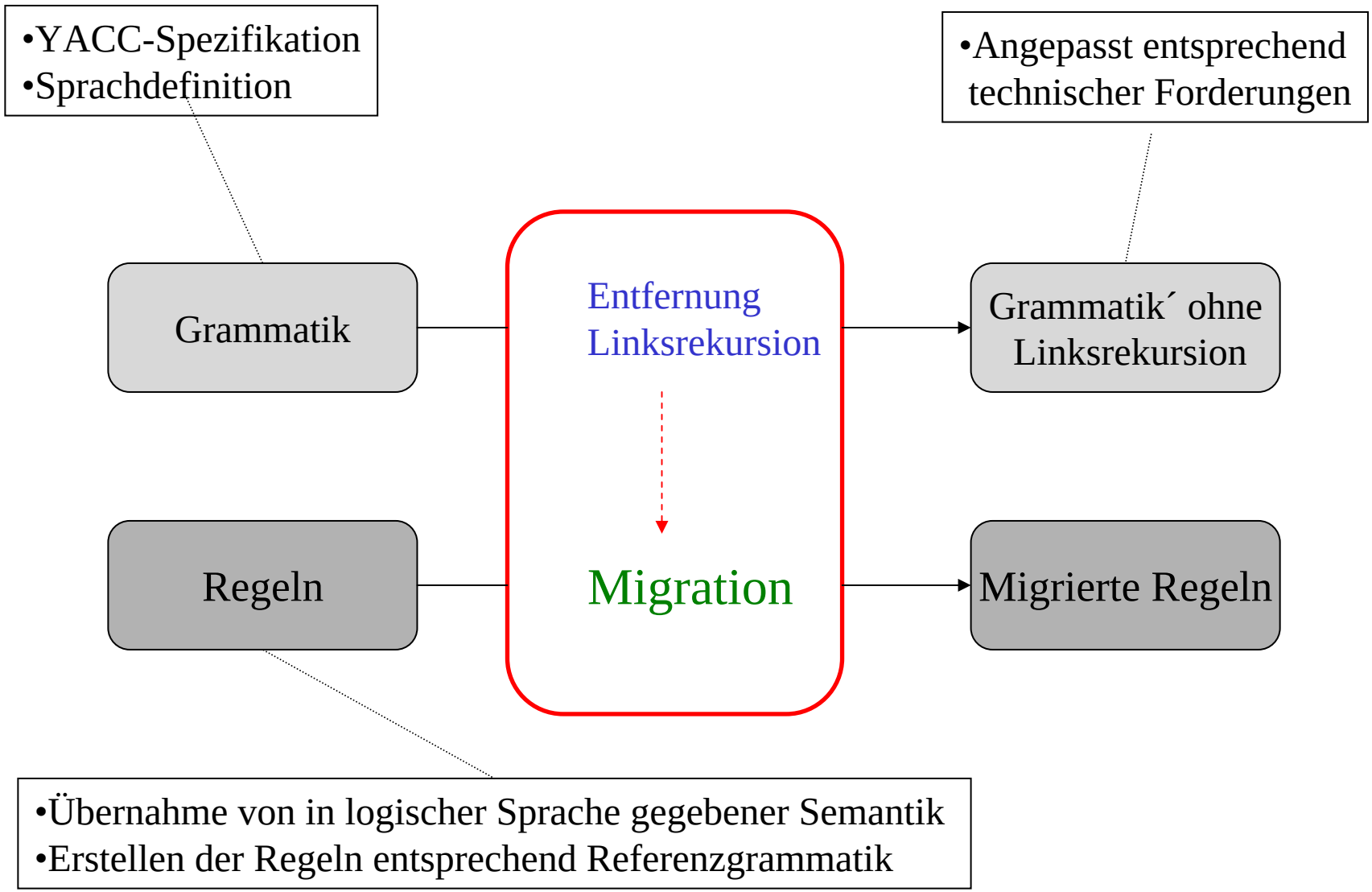
Co-Transformation...

... transformiert voneinander abhängige Softwareartefakte verschiedener Art gleichzeitig. Dabei ist die Transformation auf die zugrundeliegende Struktur ausgerichtet. [Lämmel]



Co-Transformation

Beispiel 1


•YACC-Spezifikation
•Sprachdefinition
•Angepasst entsprechend technischer Forderungen
Grammatik
Entfernung Linksrekursion
Grammatik´ ohne Linksrekursion
Regeln
Migration
Migrierte Regeln
•Übernahme von in logischer Sprache gegebener Semantik
•Erstellen der Regeln entsprechend Referenzgrammatik

Entfernung der Linksrekursion

Bekannter Algorithmus:

In: $G = (V_N, V_T, P, S)$, let $V_N = \{A^1, .., A^N\}$

Out: $G' = (V'_N, V'_T, P', S)$ without left recursion

for i := 1 to N do

replace

$$A^i \rightarrow A^i \alpha_1 | \, .. \, | \, A^i \alpha_n | \beta_1 \, | \, .. | \beta_m$$

by

$$A^i \rightarrow \beta_1 B^i | \, .. | \beta_m B^i \cup$$

$$B^i \rightarrow \alpha_1 B^i \, | \, .. | \, \alpha_n B^i \, | \, \varepsilon$$

Entfernung der Linksrekursion

Bekannter Algorithmus:

In: $G = (V_N, V_T, P, S)$, let $V_N = \{A^1, .., A^N\}$

Out: $G' = (V'_N, V'_T, P', S)$ without left recursion

for i := 1 to N do

replace

$$A^i \rightarrow A^i \alpha_1 | \, .. \, | \, A^i \alpha_n | \beta_1 \, | \, .. | \beta_m$$

by

$$A^i \rightarrow \beta_1 B^i | \, .. | \beta_m B^i \; \cup$$
$$B^i \rightarrow \alpha_1 B^i \, | \, .. | \, \alpha_n B^i \, | \, \varepsilon$$

Problem:

Grammatik komplex!
Originale Sprache?

Entfernung der Linksrekursion

Bekannter Algorithmus:

In: $G = (V_N, V_T, P, S)$, let $V_N = \{A^1, .., A^N\}$

Out: $G' = (V'_N, V'_T, P', S)$ without left recursion

for i := 1 to N do

replace

$$A^i \rightarrow A^i \alpha_1 | \ .. \ | A^i \alpha_n | \beta_1 | \ .. | \beta_m$$

{Semantische Regeln für A^i}

by

$$A^i \rightarrow \beta_1 B^i | \ .. | \beta_m B^i \ \cup$$

$$B^i \rightarrow \alpha_1 B^i | \ .. | \alpha_n B^i | \ \varepsilon$$

{??????????????????}

Problem:

Grammatik komplex!
Originale Sprache?
Semantische Regeln?

Idee am Beispiel:
arithmetischer Ausdruck S-AG

$E_0 \rightarrow E_1 + T$	$\{ E_0.v := E_1.v + T.v\}$
$\mid E_1 - T$	$\{ E_0.v := E_1.v - T.v\}$
$\mid T$	$\{ E_0.v := T.v \}$
$T_0 \rightarrow T_1 * F$	$\{ T_0.v := T_1.v * F.v\}$
$\mid T_1 / F$	$\{ T_0.v := T_1.v / F.v\}$
$\mid F$	$\{ T_0.v := F.v \}$
$F \rightarrow N$	$\{ F.v := N.v \}$
$\mid (E)$	$\{ F.v := E.v \}$

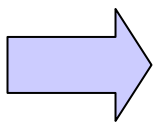
Idee am Beispiel:
Ausdruck AG ohne Linksrekursion

$E_0 \rightarrow E_1 + T$
$\mid E_1 - T$
$\mid T$

$T_0 \rightarrow T_1 * F$
$\mid T_1 / F$
$\mid F$

$F \rightarrow N$
$\mid (E)$

LRR



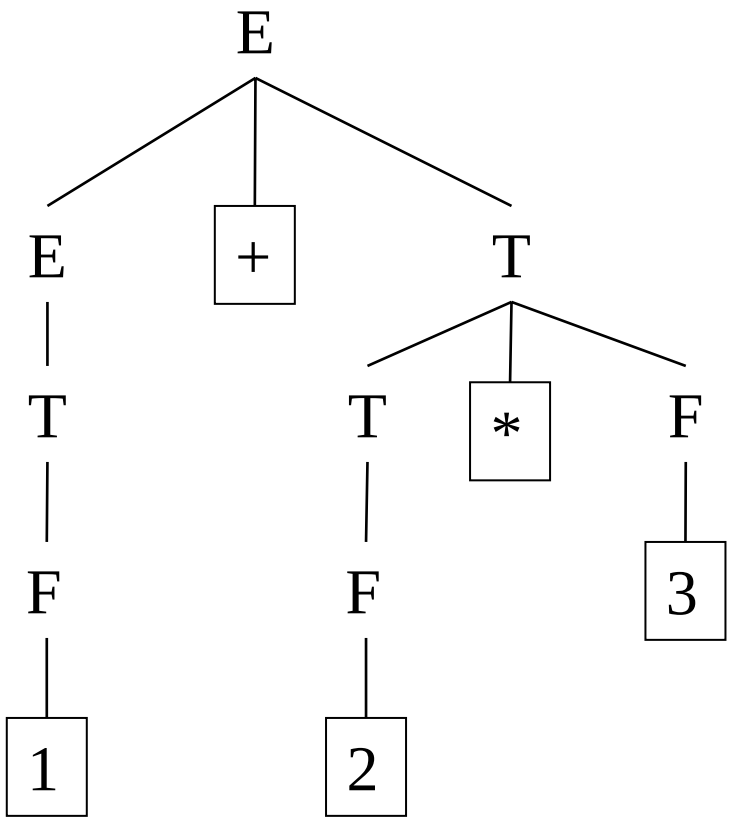
$E \rightarrow T\ E'$
$E' \rightarrow + T\ E'$
$\mid\!- T\ E'$
$\mid \varepsilon$

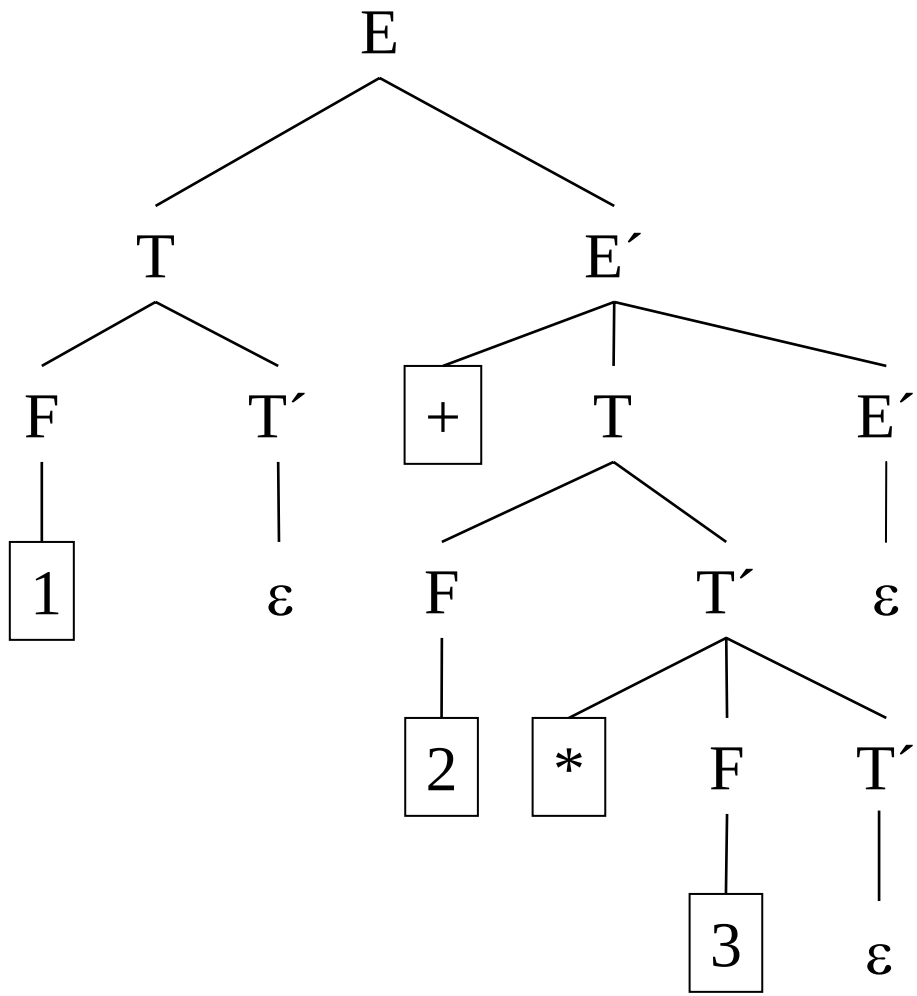
$T \rightarrow F\ T'$
$T' \rightarrow * F\ T'$
$\mid / F\ T'$
$\mid \varepsilon$

$F \rightarrow N$
$\mid (E)$

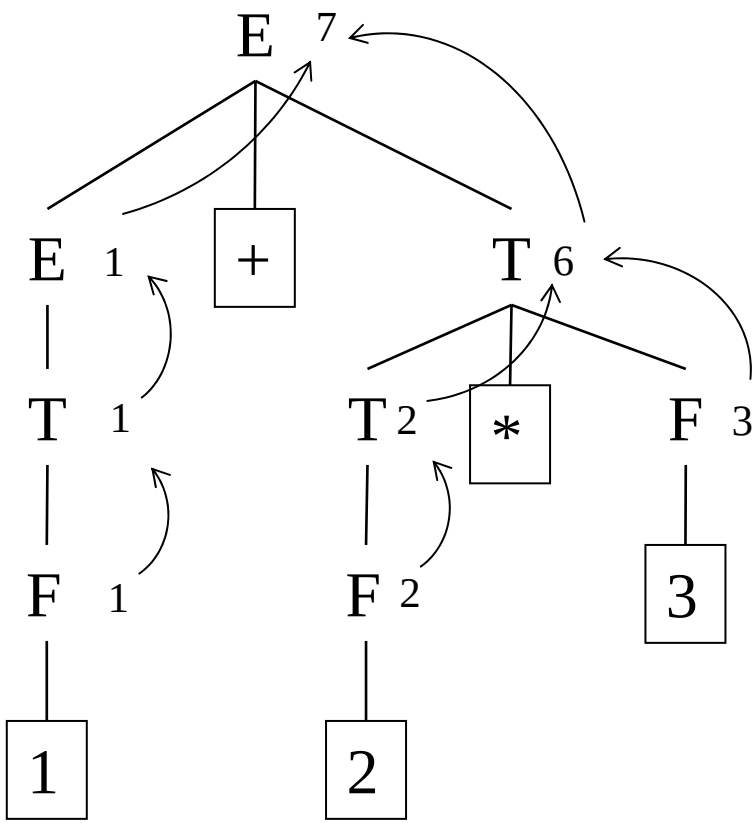
Idee am Beispiel:

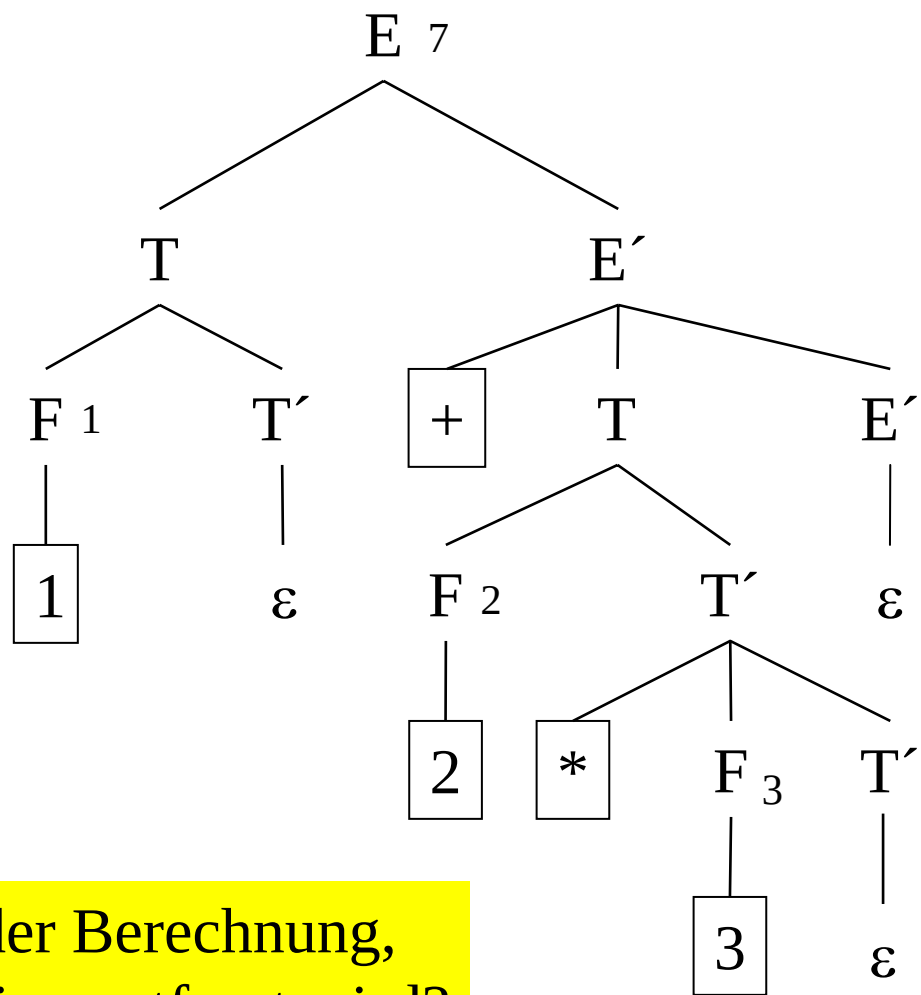
Ableitungsbaum für 1+2*3





Idee am Beispiel:
Ableitungsbaum für 1+2*3

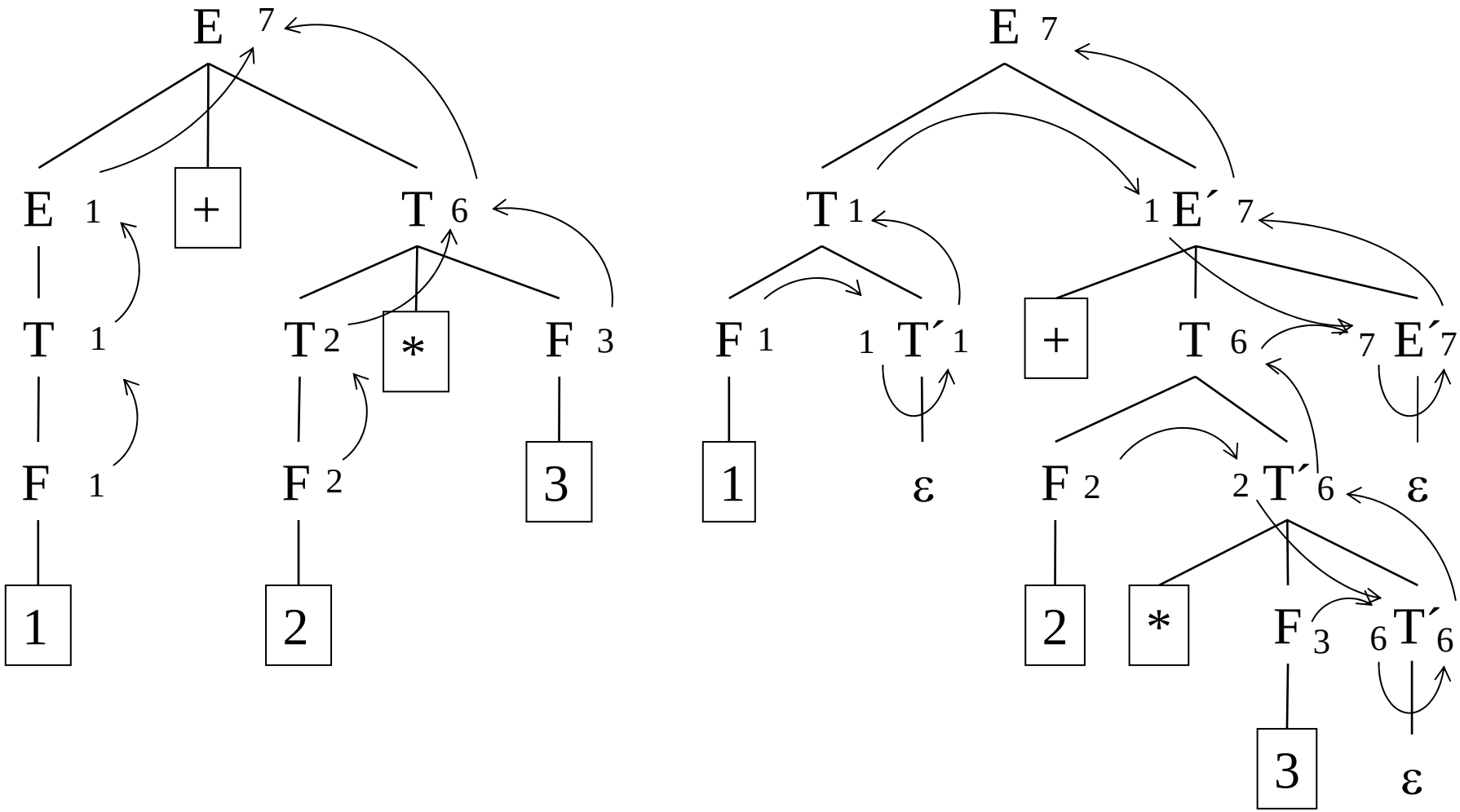




Was passiert mit der Berechnung, wenn Linksrekursion entfernt wird?

Idee am Beispiel:

Ableitungsbaum für 1+2*3



Idee am Beispiel:

Migrierte Regeln für das Beispiel

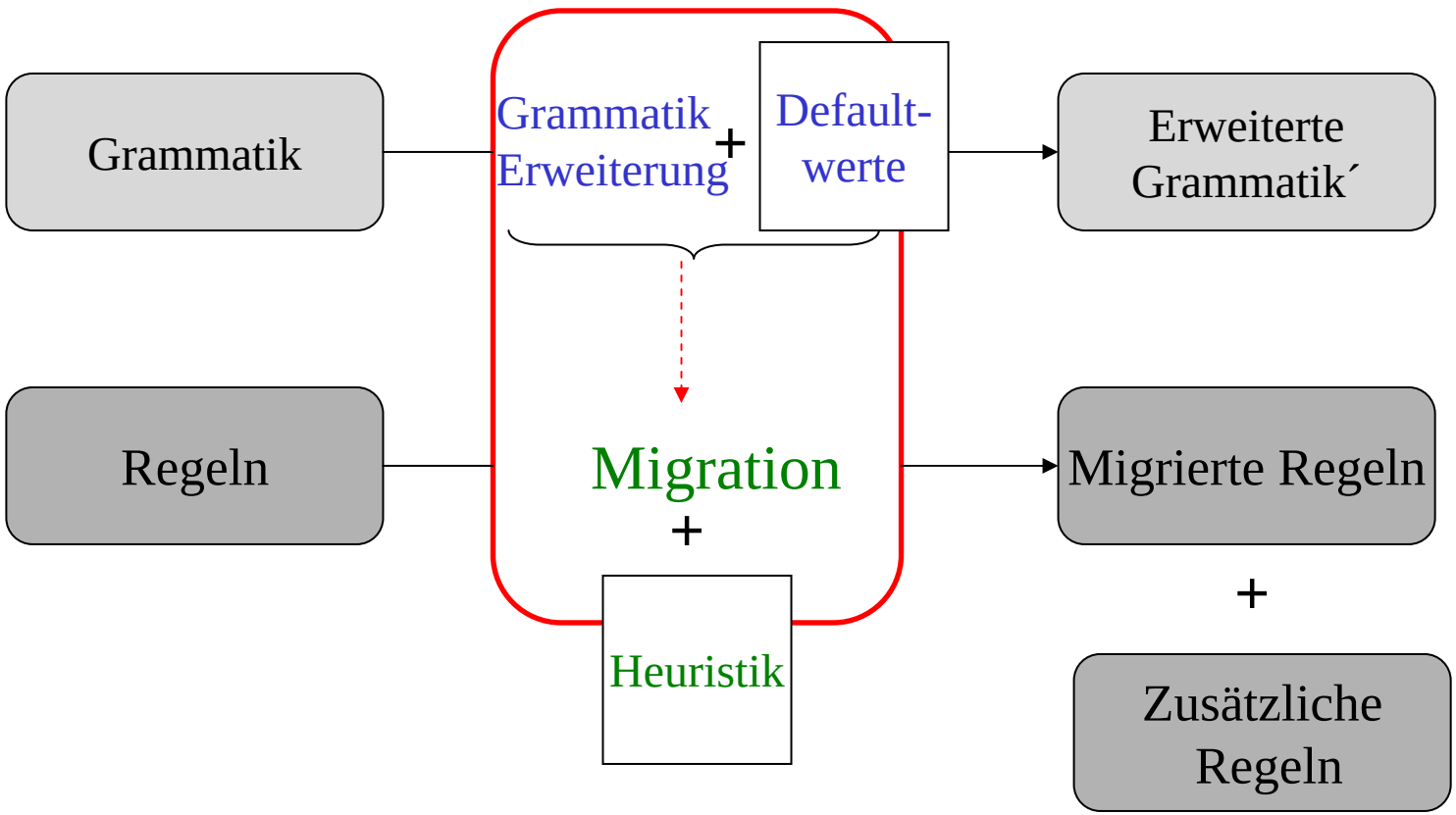
$E \rightarrow T\ E'$	{ E´.i := T.v, E.v := E´.v }
$E'_0 \rightarrow + T\ E'_1$	{ E'_1.i := E'_0.i + T.v, E'_0.v := E'_1.v }
|– T E'_1	{ E'_1.i := E'_0.i - T.v, E'_0.v := E'_1.v }
| ε	{ E'_0.v := E'_0.i }
$T \rightarrow F\ T'$	{ T´.i := F.v, T.v := T´.v }
$T'_0 \rightarrow * F\ T'_1$	{ T'_1.i := T'_0.i * F.v, T'_0.v := T'_1.v }
| / F T'_1	{ T'_1.i := T'_0.i / F.v, T'_0.v := T'_1.v }
| ε	{ T'_0.v := T'_0.i }
$F \rightarrow N$	{ F.v := N.v }
| (E)	{ F.v := E.v }

Es existiert eine allgemeine Abbildung!

Ergebnis 1

- Verwendung einer ε-freien linksrekursiven Grammatik auch für Top-Down-basierte Werkzeuge
- Also auch: Verringerung der Entfernung der genutzten Grammatik von der Vorlage
- Verbesserung bei der Lesbarkeit (eben weil nicht mehr die Komplexität durch LRR)

Beispiel 2: Layouterhaltung nachträglich



```
if E(i,x) then
  while E2(n,x) do
     i := f(m,x);
     if E(n,m) then
       something;
     endif
     n := n - i;
  od -- some comment
endif
```

```
if E(i,x) then
  i := f(m,x);
  while E2(n,x) do
     if E(n,m) then
       something;
     endif
     n := n - i;
  od -- some comment
endif
```

Mit Prolog (ohne PrettyPrinting):

```
if E ( i , x ) then i := f ( m , x ) ; while
E2 ( n , x ) do if E ( n , m ) then something
; endif n := n - i; od endif
```

Transformation mit Prolog

```
loop_op( stmts(if(EXP,Ss), Ss2), stmts(if(EXP,NewSs),N2) ):-
   loop_op(Ss,NewSs),loop_op(Ss2, N2).
```

Traversiere IF

```
loop_op(
   stmts(while(EXP,stmts(assign(ID,EXP2),Ss)),Ss2) ,
   stmts(assign(ID,EXP2),stmts(while(EXP,NewSs),N2)) ) :-
   changes(ID,Ss,0),loop_op(Ss,NewSs), loop_op(Ss2,N2).
```

Vertausche statements

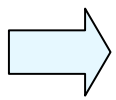
```
changes(ID, stmts(if(EXP, Ss),Ss2),Cs) :-
   changes(ID,Ss,Cs1), changes(ID,Ss2,Cs2), Cs is Cs1 +Cs2.

changes(ID, stmts(while(EXP,Ss)),Cs) ) :- changes(ID,Ss,Cs).
```

Test auf ID

Übliche Lösung

- Grammatikerweiterung um Nichtterminal
  `lay` für die Darstellung von Layout

```
stmt :
 “if“ exp
     “then“ stmts
     “endif“ {if}
```



```
stmt :
 lay “if“ exp lay
     “then“ stmts
 lay “endif“ {if}
```

- Müssen keine Sprachkonstrukte sein, einfach Strukturbeschreibungen von Daten

– Transformationsregeln können jetzt Layout (-knoten eines Termes) berücksichtigen

```
loop_op(stmts(if(L,EXP,L2,Ss,L3), Ss2),
        stmts(if(L,EXP,L2,NewSs,L3),NewSs2))
:- loop_op( Ss,NewSs ),loop_op( Ss2,NewSs2).
```

– Volle Kontrolle über Layoutbehandlung

– Problem:

1. Unnötige neue Komplexität
   a) Auf Grammatikebene
   b) in Transformationregeln
2. Was ist mit existierenden Transformations-regeln?

Lösungsansatz

- Grammatikerweiterungen (hier durch Layout) in einer Trace dokumentieren
- Generieren der internen Darstellung
- Anpassung der Muster in Regeln entsprechend der Trace und einer Heuristik
  - Im einfachsten Fall ein einfaches Zusammenweben von Variablen/Defaultwerten und Teilen von Mustern
- Schwierigkeit ist nicht das Weben, sondern Wahl der Defaultwerte und Heuristik
- Heuristik entspricht Sinn des eingefügten Nichtterminals
- Wichtig: Änderung der Grammatik -> Änderung der Regeln

Fälle von Positionen eines neu konstruierten Terms

Fall 1 : Ausgabeposition der LHS

a) Funktor des Terms / Subterm einer Ein-/einer Ausgabeposition auf RHS

f(r(**L**,a...) ,..,r(**L**, h,...)) :-
f_i(b(...), .., c(...)),
f_n(d(...), .., e(...)).

Wiederverwendung gebundener Information

b) Funktor erscheint nicht auf Eingabeposition der LHS oder Ausgabeposition der RHS

f(a(...), .., r(**“_“** ,..,...)) :-
f_i(b(...), .., c(...)),
f_n(d(...), .., e(...)).

Default-Werte

c) Es gibt verschiedene Terme, die mit dem gleichen Funktor passen wie der konstruierte Term (Intention nicht klar)

f(r(**L**,a...),r(**L_2**,f...),r(**L?**, h,...)) :-
f_i(b(...), .., c(...)),
f_n(d(...), .., e(...)).

Erstes Auftreten oder Default-Werte (Heuristik)

Fallunterscheidung II

Fall 2 : Eingabeposition in Prädikat auf RHS

a) Funktor vom Term/ Teilterm einer Eingabeposition auf LHS oder einer Ausgabeposition auf RHS (bzgl. DF)

Nutze gebundenes Layout

f(r(**L** ,a...) , .., R) :-
  f_i(r(**L** ,...) , .., R),
  f_n(d(...), .., e(...)).

b) Funktor kommt nicht auf Eingabeposition der LHS und nicht auf Ausgabeposition auf RHS vor

Default-Werte

f(a(...), .., R) :-
  f_i(r(“_“ ,...) , .., R),
  f_n(d(...), .., e(...)).

c) Mehrfachauftreten

Erstes Auftreten oder Default-Werte (Heuristik)

f(r(**L**,a...), r(**L_2** ,..) ,R) :-
  f_i(r(**L?** ,...) , .., R),
  f_n(d(...), .., e(...)).

Fall 3 : Ausgabepositionen auf RHS

Neue Variablen

f(a(...),.., ...) :-
  f_i(...,r(**L_i** ,d(..))),
  f_n(d(...), r(**L_j** ,..)).

Vorgehen

- Traversierung über Transformationsregeln (mit Trace-Information)
  1. Initialisierung des Webens
  2. Bestimmen der Trace für den aktuellen Knoten der AS
  3. Bestimme wiederverwendbares Layout mittels Heuristik
  4. Weben der originalen Terme, Default-Werte, neuer Variablen für Layout, oder Variablen für Terme entsprechend der Heuristik

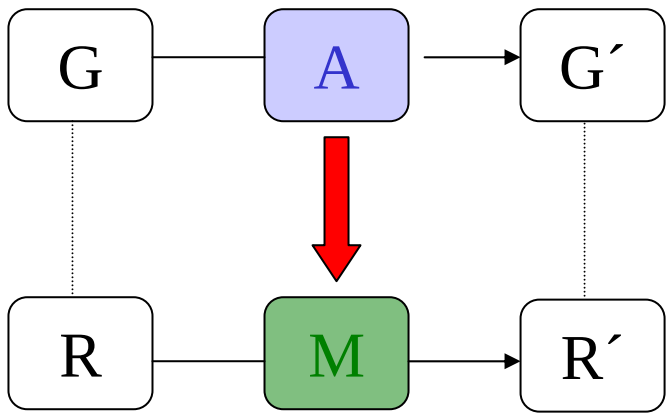
Ergebnis 2

- Verbesserung der Grammatik
  - Arten von Kommentaren
  - Eingebettete Anweisungen
- Wiederverwendung alter Regeln
- Weiterarbeiten im alten Stil möglich, Erweiterung wird vor dem Nutzer verborgen (Vereinfachung für Erstellen von Transformationsregeln)
- Spezifikation für Teilmuster, die für eine Aufgabe interessant ist (entspricht einer Agile Parsing-Methode)

Abschließende Bemerkung

Automatisch Migration von semantischen Regeln während Entfernung der Linksrekursion in AGs

so speziell, dass es sich nicht einfach durch Teiländerungen beschreiben lässt.



Grammatikerweiterung mit automatischer Migration von Transformationsregeln, Mustervereinfachung

zu allgemein, dadurch die Schwierigkeit mit Default-Werten und Heuristik

- Transformation von AG zu SAG
- Entfernung von ε-Regeln
- Gibt es mehr (sinnvolle) Co-Transformationen Grammatikanpassungen?

EOF

wlohmann@informatik.uni-rostock.de