Verallgemeinerte Tabellenparser in Prolog

Grammatikformat: bedingte DCG-Regeln, vorberechnete Prädikate

`del/1`	für	$\{ A \mid A \Rightarrow^+ \epsilon \}$,
`chain/2`,	für	$\{ (A, B) \mid A \Rightarrow^+ B \}$
`first/2`	für	$\{ (B, A) \mid \exists\gamma \in (Cat \cup \Sigma)^* \ A \Rightarrow^+ B\gamma \}$
`predicted`	für	$predict_{ep}$ bzw. $predict_{lc}$

$Wald(\alpha, a_{i+1} \cdots a_j)$ sei die Menge Analysen zu $\alpha \Rightarrow^* a_{i+1} \cdots a_j$, d.h. der Baumfolgen $\vec{t}$, deren Wurzeln mit α und deren Blätter mit $a_{i+1} \cdots a_j$ markiert sind.

Statt der Kanten $i \overset{(A\to\alpha\cdot\beta)}{\longrightarrow} j$ bilden wir Kanten $i \overset{A\to\vec{t}\cdot\beta}{\longrightarrow} j$ mit

$$(A \to \alpha \cdot \beta) \in \dot{P} \quad \text{und} \quad \vec{t} \in Wald(\alpha, a_{i+1} \cdots a_j),$$

die zu einer Top-Down-Erwartungskategorie S passen.

Die *partiellen* Analysen $i \overset{A\to\vec{t}\cdot\beta}{\longrightarrow} j$, bei denen $\beta \neq \epsilon$ ist, sammelt

$$\texttt{ptrees}(\texttt{I}, \texttt{J}, \texttt{Kategorie}, \texttt{Baumliste}, \texttt{Beta}).$$

Die *vollständigen* Analysen $i \overset{A\to\vec{t}\cdot}{\longrightarrow} j$, bei denen $\beta = \epsilon$, sammelt

$$\texttt{ctrees}(\texttt{I}, \texttt{J}, \texttt{Kategorie}, \texttt{Baum}).$$

Jeder Baum $t = A(\vec{t})$ besteht aus der Wurzel A und den direkten Teilbäumen $\vec{t}$. (Es wird nicht nach Regeln $A \to \alpha$ getrennt.)

Verallgemeinerter Earley-Parser in Prolog

⟨*Verallgemeinerter Earley-Parser*⟩≡

```
:- ensure_loaded('Parser/main.pl').
:- dynamic inputlaenge/1, expect/2, ctrees/4, ptrees/5.
⟨Earley-Pratt-Leiß⟩
⟨Lexikalische Vorausschau⟩
⟨Hilfsfunktionen⟩
```

Der Hauptaufruf ist wieder `parse(Wortliste+,Baum?)`:

⟨*Earley-Pratt-Leiß*⟩≡

```
parse([],_) :-
    !,write('Fehler: Leere Eingabe.\n'),fail.
parse([W|Ws],Baum) :-
    retractall(ctrees(_,_,_,_)),
    retractall(ptrees(_,_,_,_,_)),
    retractall(expect(_,_)),
    retractall(inputlaenge(_)),
    length([W|Ws],N),
    assert(inputlaenge(N)), % s. complete2
    earley([W|Ws],0),
    !,extract(0,N,Baum).
```

Durch `earley/2` wird die Tabelle aufgebaut; da sie nur kanonische Bäume sammelt, muß zum Schluß eine Kette von einem Startsymbol zur Wurzel ergänzt werden:

⟨*extract*⟩≡

```
extract(I,J,Baum) :-
        start_symbol(S),
        ctrees(I,J,Cat,Tree),
        (chain(S,Cat) -> Baum = [S,Tree]
        ; S = Cat, Baum = Tree).
```

Die Hauptschleife, `earley/2`, liest die Wortfolge der Eingabe von links nach rechts.

⟨*Earley-Pratt-Leiß*⟩+≡
```
earley([W|Ws],J) :-
        M is J + 1,
        scan(J,M,W),
        complete(J),
        predict(J),
        earley(Ws,M).
earley([],J) :-
        !,complete(J).
```

Für jedes Eingabewort werden seine lexikalischen Lesarten zu Wurzeln entsprechender Analysebäume:

⟨*Earley-Pratt-Leiß*⟩+≡
```
scan(I,J,W) :-
        lexword(I,W,Cat),
        add(ctrees(I,J,Cat,[Cat,W])),
        fail.
scan(_,_,_).
```

Um bei j „brauchbare“ Regeln vorauszusagen, sucht man aus den vorberechneten brauchbaren Regeln unter `predicted/6` die aus, die zu einer für $a_{j+1} \cdots a_n$ erwarteten Kategorie `C` und einer Kategorie `Cat` von a_{j+1} passen.[4]

[4] In `predicted/6` wird sowohl die Baumfolge `AlphaTrees` als auch deren Wurzelfolge `Alpha` vermerkt; für den Parser brauchen wir nur die Baumfolge, für den Erkenner nur die Wurzeln.

Predictor

Löschungen `AlphaTrees` vereinfachen wir im Folgenden zu `[]`.

$$\frac{j \cdot C, \quad (A \to \alpha \cdot B\beta) \in predict_{ep}(C, a_{j+1})}{j \overset{A \to \epsilon \cdot B\beta}{\longrightarrow} j} \quad (Predict)$$

⟨*Earley-Pratt-Leiß*⟩+≡

```
predict(J) :-
        (ein(lookaheadLexCats)
        -> inputLexCats(J,InputLexCats) ; true),
        nextword_cat(J,LexCat),
        (J = 0 -> expect_initial(LexCat) ; true),
        expect(J,C),
        predicted(C,LexCat,A,_Alpha,[B|Beta],AlphaTrees),
        (ein(lookaheadLexCats)
        -> memberLexCats([B|Beta],InputLexCats) ; true),
        add(ptrees(J,J,A,[],[B|Beta]))
        addExpect(J,B),     % (!)
        fail.
predict(_).
```

An der Anfangsposition erwartet man nur Startkategorien, die mit einer Kategorie `Cat` des ersten Wortes verträglich sind:

⟨*Earley-Pratt-Leiß*⟩+≡

```
expect_initial(Cat) :-
        start_symbol(C),
        first(Cat,C),           % ; C = Cat
        addExpect(0,C),
        fail.
expect_initial(_).
```

Lexikalische Kategorien-Vorausschau

`lookaheadLexCats` beseitigt vorausgesagte Regeln ($A \to \alpha \cdot \beta$), deren β eine (obligatorische) lexikalische Kategorie enthält, die im Lexikon der Resteingabe nicht vorkommt.

Dazu berechnet man bei jeder Position j die Kategorienmenge

⟨*Lexikalische Vorausschau*⟩≡

```
inputLexCats(J,LexCats) :-
        inputLexCats(J,[],LexCats).
inputLexCats(J,Acc,Cats) :-
        inputlaenge(N),
        setof(C,Word^lexword(J,Word,C),Cs),
        append(Cs,Acc,CsAcc),
        K is J+1,
        (K < N -> inputLexCats(K,CsAcc,Cats) ; Cats = CsAcc).
```

und testet, ob die Kategorien der Regeln darin vorkommen:

⟨*Lexikalische Vorausschau*⟩+≡

```
memberLexCats([(A;B)|As],LexCats) :-
        !, (memberLexCats([A],LexCats)
           ; memberLexCats([B],LexCats)),
        memberLexCats(As,LexCats).
memberLexCats([(A,B)|As],LexCats) :-
        !, memberLexCats([A,B|As],LexCats).
memberLexCats([?(A)|As],LexCats) :-
        !, memberLexCats(As,LexCats).
memberLexCats([{Goal}|As],LexCats) :-
        !, memberLexCats(As,LexCats).
memberLexCats([A|As],LexCats) :-
        (lexCat(A) -> \+ \+ memberchk(A,LexCats) ; true),
        memberLexCats(As,LexCats).
memberLexCats([],_).
```

Da u.a. alle Kopf-Komplement-Regeln eine lexikalische Kategorie enthalten, ist der Effekt erheblich (Faktor 5 bei `es.grm`)

Completor-Schleife

Ist a_{j+1} gelesen, so geht man zurück zu Positionen $i \leq j$ und vervollständigt die Analysen von $a_{i+1} \cdots a_j$, indem man die von $a_{i+1} \cdots a_k$ mit denen von $a_{k+1} \cdots a_j$ kombiniert, für jedes $k = j, j-1, \ldots, i$.

$\langle$*Earley-Pratt-Leiß*$\rangle{+}{\equiv}$

```
complete(J) :-
        (0 < J -> I is J -1, complete(I,J) ; true).
complete(I,J) :-
        K is J - 1,
        complete(I,K,J),
        (0 < I  -> M is I - 1, complete(M,J) ; true).
complete(I,K,J) :-  % 0 =< I und K < J garantiert
        (I < K
        -> complete1(I,K,J), complete2(I,K,J),
            M is K - 1,
            complete(I,M,J)
        ; true),
        (I = K -> complete2(I,K,J) ; true).
```

Zur Erweiterung der partiellen Analyse $i \xrightarrow{A \to \vec{t} \cdot B\beta} j$ mit einer vollständigen Analyse $k \xrightarrow{s} j$ unterscheidet man zwei Fälle:

Complete-1

(i) β enthält nur löschbare Kategorien:

$$\frac{i \xrightarrow{A \to \vec{t} \cdot B\beta} k, \quad B \Rightarrow^* D, \quad k \xrightarrow{D \to \vec{s} \cdot} j, \quad \beta \Rightarrow^* \epsilon}{i \xrightarrow{A \to \vec{t} r \cdot} j} \quad (Complete_1)$$

wobei $r := \begin{cases} s := D(\vec{s}), & \text{falls } B \equiv D, \\ B(s), & \text{sonst.} \end{cases}$

Man konstruiert eine *vollständige* Analyse von $a_{i+1} \cdots a_j$.

$\langle$*Earley-Pratt-Leiß*$\rangle+\equiv$

```
complete1(I,K,J) :-
    ctrees(K,J,D,DTree),
    ptrees(I,K,A,AlphaTrees,[B|Beta]),
    connect(B,D,DTree,BTree),
    deletables(Beta),
    BetaTrees = [],
    append(AlphaTrees,[BTree|BetaTrees],Subtrees),
    add(ctrees(I,J,A,[A|Subtrees])),
    fail.
complete1(_,_,_).
```

Da dies nur für $i < k < j$ gemacht wird, verzweigt die neue Analyse von $w(i, j)$ an der Wurzel.

(Es ist effizienter, erst die `ctrees(K,J,D,DTree)` und dann die `ptrees(I,K,...)` zu suchen, als umgekehrt.)

Complete-2

(ii) β enthält mindestens eine Kategorie C, die mit dem nächsten Eingabesymbol a_{j+1} verträglich ist:

$$\frac{i \overset{A\to\vec{t}\cdot B\vec{B}C\gamma}{\longrightarrow} k, \quad B \Rightarrow^* D, \quad k \overset{D\to\vec{s}\cdot}{\longrightarrow} j, \qquad \vec{B} \Rightarrow^* \epsilon, \qquad a_{j+1} <^*_{lc} C}{i \overset{A\to\vec{t}r\cdot C\gamma}{\longrightarrow} j} \quad (Complete_2)$$

wobei $r := \begin{cases} s := D(\vec{s}), & \text{falls } B \equiv D, \\ B(s), & \text{sonst.} \end{cases}$

Man konstruiert eine *partielle* Analyse von $a_{i+1} \cdots a_j$.

⟨*Earley-Pratt-Leiß*⟩+≡

```
complete2(I,K,J) :-
    inputlaenge(N),  % C erzeugt mindestens a_i+1
    J < N,
    ctrees(K,J,D,DTree),
    ptrees(I,K,A,AlphaTrees,[B|Beta]),
    connect(B,D,DTree,BTree),
    append(Bs,[C|Gamma],Beta),
    deletables(Bs),
    nextword_cat(J,LexCat),
    leftcorner_rtc(LexCat,C),
    BsTrees = [],
    append(AlphaTrees,[BTree|BsTrees],AlphaBBsTrees),
    add(ptrees(I,J,A,AlphaBBsTrees,[C|Gamma])),
    addExpect(J,C),          % (!)
    fail.
complete2(_,_,_).
```

Berechnung des Earley-Predictors

Mit früher angegebenen Hilfsfunktionen berechnet man

$$predict_{ep}(E, c) := \{(A \to \alpha \cdot B\beta) \in \dot{P} \mid A <^*_{lc} E,\ \alpha \Rightarrow^* \epsilon,\ c <^*_{lc} B\}$$

wie folgt:

⟨*Vorberechnung brauchbarer Regeln*⟩≡

```
predict_static(E,LexCat) :-
    setof((A,Alpha,[B,B2|Beta],AlphaTrees),
           Alphas^As^((A=E ; first(A,E)),
           (A --> Alphas),
           tupleToList(Alphas,As),
           append(Alpha,[B,B2|Beta],As),
           deletables(Alpha),
           leftcorner_rtc(LexCat,B),
           canonicalTrees(Alpha,AlphaTrees)),
           Rules),
    add_predicted(E,LexCat,Rules).

add_predicted(E,LexCat,
       [(A,Alpha,[B,B2|Beta],AlphaTrees)|Rules]) :-
    add(predicted(E,LexCat,A,Alpha,[B,B2|Beta],
                  AlphaTrees)),
    add_predicted(E,LexCat,Rules).
add_predicted(_E,_LexCat,[]).
```

Die Berechnung des Leftcorner-Predictors geht analog.

Beispiel

⟨*dcg.4.grm*⟩≡

```
auxiliary_predicates([start_symbol/1]).

% Startsymbole:
start_symbol(np).    start_symbol(ap).

% Regeln:
:- op(1100,xfx,'-->').
np --> ?det, ?ap, n, ?rel.
ap --> pp(P), a(P).
ap --> a.
pp(P) --> pr(P), np.
rel --> rpro, np, tv(n,n).
rel --> rpro, pp(P), tv(n,pr(P)).
rel --> rpro, iv.

?(C) --> [].        % ?(C) --> C. % implizit

% Lexikon:
word(a,a).               word(d,det).
word(a,a(p1)).           word(n,n).
word(a,a(p2)).           word(r,rpro).
word(p,pr(p1)).          word(p,pr(p2)).

word(v,iv).              word(v,tv(n,n)).
word(w,tv(n,pr(p1))).  word(w,tv(n,pr(p2))).

% Beispiele:
% d p n a n r n v.
% d p n a n r p n w.
```

Vorberechnete Hilfstabellen

⟨*dcg.4.grm Hilfstabellen (Earley und Leftcorner)*⟩≡

```
:-op(1100, xfx, (-->)).

start_symbol(np).   start_symbol(ap).

/*****       Deletable categories        ******/

:- dynamic del/1.
del(?A). % ...

/*****             Chain rules           ******/

:- dynamic chain/2.
chain(np, n).      chain(ap, a).

/*****            First-relation         ******/

:- dynamic first/2.
first(det, np).    first(ap, np).
first(n, np).      first(pp(A), ap).
first(a, ap).      first(pr(A), pp(A)).
first(rpro, rel).  first(pr(A), np).
first(pp(A), np).  first(a, np).
first(pr(A), ap).
```

Predictor (explizit) für den Earley-Parser

⟨*dcg.4.grm.epl.pl*⟩≡

```
/*** predicted(C,LexC,A,Alpha,[B|Beta],Tree) **/
:- dynamic predicted/6.

predicted(np, det,
     np, [], [?det, ?ap, n, ?rel], []).
predicted(np, n,
     np, [?det, ?ap], [n, ?rel],
     [[?det, []], [?ap, []]]).
predicted(pp(A), pr(A),
     pp(A), [], [pr(A), np], []).
predicted(rel, rpro,
     rel, [], [rpro, iv], []).
predicted(rel, rpro,
     rel, [], [rpro, np, tv(n, n)], []).
predicted(rel, rpro,
     rel, [], [rpro, pp(A), tv(n, pr(A))], []).
predicted(np, pr(A),
     ap, [], [pp(A), a(A)], []).
predicted(np, pr(A),
     np, [?det], [?ap, n, ?rel], [[?det, []]]).
predicted(np, pr(A),
     pp(A), [], [pr(A), np], []).
predicted(np, a,
     np, [?det], [?ap, n, ?rel], [[?det, []]]).
predicted(ap, pr(A),
     ap, [], [pp(A), a(A)], []).
predicted(ap, pr(A),
     pp(A), [], [pr(A), np], []).
/*****          End                  ******/
```

Analyse mit .epl-Parser

⟨*Trace*⟩≡

```
?- parse. d p n a n r n v.

scan(0,1,d)
add: ctrees(0, 1, det, [det, d])
predict(0)
add: expect(0, np)
add: ptrees(0, 0, np, [], [?det, ?ap, n, ?rel])
add: expect(0, det)
scan(1,2,p)
add: ctrees(1, 2, pr(p1), [pr(p1), p])
add: ctrees(1, 2, pr(p2), [pr(p2), p])
complete(0,0,1)
add: ptrees(0, 1, np, [[det, d]], [?ap, n, ?rel])
add: expect(1, ap)
predict(1)
add: ptrees(1, 1, ap, [], [pp(p1), a(p1)])
add: expect(1, pp(p1))
add: ptrees(1, 1, pp(p1), [], [pr(p1), np])
add: expect(1, pr(p1))
add: ptrees(1, 1, ap, [], [pp(p2), a(p2)])
add: expect(1, pp(p2))
add: ptrees(1, 1, pp(p2), [], [pr(p2), np])
add: expect(1, pr(p2))
scan(2,3,n)
add: ctrees(2, 3, n, [n, n])
complete(1,1,2)
add: ptrees(1, 2, pp(p1), [[pr(p1), p]], [np])
add: expect(2, np)
add: ptrees(1, 2, pp(p2), [[pr(p2), p]], [np])
```

```
complete(0,1-0,2)
predict(2)
add: ptrees(2, 2, np,
            [[?det, []], [?ap, []]], [n, ?rel])
add: expect(2, n)
scan(3,4,a)
add: ctrees(3, 4, a, [a, a])
add: ctrees(3, 4, a(p1), [a(p1), a])
add: ctrees(3, 4, a(p2), [a(p2), a])
complete(2,2,3)
complete(1,2,3)
add: ctrees(1, 3, pp(p1),
            [pp(p1), [pr(p1), p], [np, [n, n]]])
add: ctrees(1, 3, pp(p2),
            [pp(p2), [pr(p2), p], [np, [n, n]]])
complete(1,1,3)
add: ptrees(1, 3, ap,
            [[pp(p1), [pr(p1), p], [np, [n, n]]]],
            [a(p1)])
add: expect(3, a(p1))
add: ptrees(1, 3, ap,
            [[pp(p2), [pr(p2), p], [np, [n, n]]]],
            [a(p2)])
add: expect(3, a(p2))
complete(0,2-0,3)
predict(3)
scan(4,5,n)
add: ctrees(4, 5, n, [n, n])
complete(3,3,4)
complete(2,3-2,4)
complete(1,3,4)
add: ctrees(1, 4, ap,
```

```
            [ap, [pp(p1), [pr(p1), p],
                  [np, [n, n]]], [a(p1), a]])
add: ctrees(1, 4, ap,
            [ap, [pp(p2), [pr(p2), p],
                  [np, [n, n]]], [a(p2), a]])
complete(1,2-1,4)
complete(0,3-2,4)
complete(0,1,4)
add: ptrees(0, 4, np,
            [[det, d], [ap, [pp(p1), [pr(p1), p],
                             [np, [n,n]]], [a(p1),a]]],
            [n, ?rel])
add: expect(4, n)
add: ptrees(0, 4, np,
            [[det, d], [ap, [pp(p2), [pr(p2), p],
                             [np, [n,n]]], [a(p2),a]]],
            [n, ?rel])
complete(0,0,4)
predict(4)
scan(5,6,r)
add: ctrees(5, 6, rpro, [rpro, r])
complete(4,4,5)
complete(3,4-3,5)
complete(2,4-2,5)
complete(1,4-1,5)
complete(0,4,5)
add: ctrees(0, 5, np,
            [np, [det, d], [ap, [pp(p1), [pr(p1), p],
                                 [np, [n, n]]],
                            [a(p1), a]],
             [n, n], [?rel, []]])
add: ctrees(0, 5, np,
```

```
            [np, [det, d], [ap, [pp(p2), [pr(p2), p],
                                 [np, [n, n]]],
                            [a(p2), a]],
                [n, n], [?rel, []]]])
add: ptrees(0, 5, np,
            [[det, d], [ap, [pp(p1), [pr(p1), p],
                             [np, [n, n]]],
                        [a(p1), a]], [n, n]],
            [?rel])
add: expect(5, rel)
add: ptrees(0, 5, np,
            [[det, d], [ap, [pp(p2), [pr(p2), p],
                             [np, [n, n]]],
                        [a(p2), a]], [n, n]],
            [?rel])
complete(0,3-0,5)
predict(5)
add: ptrees(5, 5, rel, [], [rpro, iv])
add: expect(5, rpro)
add: ptrees(5, 5, rel, [], [rpro, np, tv(n, n)])
scan(6,7,n)
add: ctrees(6, 7, n, [n, n])
complete(5,5,6)
add: ptrees(5, 6, rel, [[rpro, r]], [np, tv(n, n)])
add: expect(6, np)
complete(4,5-4,6)
complete(3,5-3,6)
complete(2,5-2,6)
complete(1,5-1,6)
complete(0,5-0,6)
predict(6)
add: ptrees(6, 6, np,
```

```
            [[?det, []], [?ap, []]], [n, ?rel])
add: expect(6, n)
scan(7,8,v)
add: ctrees(7, 8, iv, [iv, v])
add: ctrees(7, 8, tv(n, n), [tv(n, n), v])
complete(6,6,7)
complete(5,6,7)
add: ptrees(5, 7, rel,
            [[rpro, r], [np, [n, n]]], [tv(n, n)])
add: expect(7, tv(n, n))
complete(5,5,7)
complete(4,6-4,7)
complete(3,6-3,7)
complete(2,6-2,7)
complete(1,6-1,7)
complete(0,6-0,7)
predict(7)
complete(7,7,8)
complete(6,7-6,8)
complete(5,7,8)
add: ctrees(5, 8, rel,
            [rel, [rpro, r], [np, [n, n]],
             [tv(n, n), v]])
complete(5,6-5,8)
complete(4,7-4,8)
complete(3,7-3,8)
complete(2,7-2,8)
complete(1,7-1,8)
complete(0,7-5,8)
complete(0,5,8)
add: ctrees(0, 8, np,
            [np, [det, d], [ap, [pp(p1),
```

```
                                            [pr(p1), p],
                                            [np, [n, n]]],
                                       [a(p1), a]],
                      [n, n],
                      [rel, [rpro, r],
                       [np, [n, n]],
                       [tv(n, n), v]]])
add: ctrees(0, 8, np,
                     [np, [det, d], [ap, [pp(p2),
                                            [pr(p2), p],
                                            [np, [n, n]]],
                                       [a(p2), a]],
                      [n, n],
                      [rel, [rpro, r],
                       [np, [n, n]], [tv(n, n), v]]])
complete(0,4-0,8)
```

Predictor (explizit) für den Leftcorner Parser

⟨*dcg.4.grm.glc.pl*⟩≡

```
/** predicted(C,D,LexC,A,Alpha,[B|Beta],Tree) **/
predicted(np, det, a|n|pr(A),
          np, [], [?det, ?ap, n, ?rel], []).
predicted(np, ap, n,
          np, [?det], [?ap, n, ?rel], [[?det, []]]).
predicted(np, n, rpro,
          np, [?det, ?ap], [n, ?rel],
          [[?det, []], [?ap, []]]).
predicted(ap, pp(A), a(A),
          ap, [], [pp(A), a(A)], []).
predicted(pp(A), pr(A), a|det|n|pr(B),
          pp(A), [], [pr(A), np], []).
predicted(rel, rpro, a|det,
          rel, [], [rpro, np, tv(n, n)], []).
predicted(rel, rpro, iv,
          rel, [], [rpro, iv], []).
predicted(rel, rpro, n|pr(A),
          rel, [], [rpro, np, tv(n, n)], []).
predicted(rel, rpro, pr(A),
          rel, [], [rpro, pp(A), tv(n, pr(A))], []).
predicted(np, pr(A), a|det|n|pr(B),
          pp(A), [], [pr(A), np], []).
predicted(np, pp(A), a(A),
          ap, [], [pp(A), a(A)], []).
predicted(np, a, n,
          np, [?det], [?ap, n, ?rel], [[?det, []]]).
predicted(ap, pr(A), a|det|n|pr(B),
          pp(A), [], [pr(A), np], []).
/*****           End                  ******/
```

Analyse mit Leftcorner-Parser

⟨*Trace*⟩+≡

```
?- parse. d p n a n r n v.

scan(0,1,d)
add: ctrees(0, 1, det, [det, d])
add: expect(0, np)
scan(1,2,p)
add: ctrees(1, 2, pr(p1), [pr(p1), p])
add: ctrees(1, 2, pr(p2), [pr(p2), p])
complete(0,0,1)
add: ptrees(0, 1, np, [[det, d]], [?ap, n, ?rel])
add: expect(1, ap)
scan(2,3,n)
add: ctrees(2, 3, n, [n, n])
complete(1,1,2)
add: ptrees(1, 2, pp(p1), [[pr(p1), p]], [np])
add: expect(2, np)
add: ptrees(1, 2, pp(p2), [[pr(p2), p]], [np])
complete(0,1-0,2)
scan(3,4,a)
add: ctrees(3, 4, a, [a, a])
add: ctrees(3, 4, a(p1), [a(p1), a])
add: ctrees(3, 4, a(p2), [a(p2), a])
complete(2,2,3)
complete(1,2,3)
add: ctrees(1, 3, pp(p1), [pp(p1), [pr(p1), p],
                                   [np, [n, n]]])
add: ctrees(1, 3, pp(p2), [pp(p2), [pr(p2), p],
                                   [np, [n, n]]])
complete(1,1,3)
```

```
add: ptrees(1, 3, ap, [[pp(p1), [pr(p1), p],
                        [np, [n, n]]]], [a(p1)])
add: expect(3, a(p1))
add: ptrees(1, 3, ap,
            [[pp(p2), [pr(p2), p], [np, [n, n]]]],
            [a(p2)])
add: expect(3, a(p2))
complete(0,2-0,3)
scan(4,5,n)
add: ctrees(4, 5, n, [n, n])
complete(3,3,4)
complete(2,3-2,4)
complete(1,3,4)
add: ctrees(1, 4, ap, [ap, [pp(p1), [pr(p1), p],
                            [np, [n, n]]],
                       [a(p1), a]])
add: ctrees(1, 4, ap,
            [ap, [pp(p2), [pr(p2), p], [np, [n, n]]],
             [a(p2), a]])
complete(1,2-1,4)
complete(0,3-2,4)
complete(0,1,4)
add: ptrees(0, 4, np,
            [[det, d], [ap, [pp(p1), [pr(p1), p],
                             [np, [n, n]]],
                        [a(p1), a]]],
            [n, ?rel])
add: expect(4, n)
add: ptrees(0, 4, np,
            [[det, d],
             [ap, [pp(p2), [pr(p2), p], [np, [n, n]]],
              [a(p2), a]]],
```

```
            [n, ?rel])
complete(0,0,4)
scan(5,6,r)
add: ctrees(5, 6, rpro, [rpro, r])
complete(4,4,5)
complete(3,4-3,5)
complete(2,4-2,5)
complete(1,4-1,5)
complete(0,4,5)
add: ctrees(0, 5, np,
            [np, [det, d], [ap, [pp(p1), [pr(p1), p],
                                         [np, [n, n]]],
                                    [a(p1), a]],
             [n, n]])
add: ctrees(0, 5, np,
            [np, [det, d], [ap, [pp(p2), [pr(p2), p],
                                         [np, [n, n]]],
                                    [a(p2), a]],
             [n, n]])
add: ptrees(0, 5, np,
            [[det, d], [ap, [pp(p1), [pr(p1), p],
                                     [np, [n, n]]],
                                [a(p1), a]], [n, n]],
            [?rel])
add: expect(5, rel)
add: ptrees(0, 5, np,
            [[det, d], [ap, [pp(p2), [pr(p2), p],
                                     [np, [n, n]]],
                                [a(p2), a]], [n, n]],
            [?rel])
complete(0,3-0,5)
scan(6,7,n)
```

```
add: ctrees(6, 7, n, [n, n])
complete(5,5,6)
add: ptrees(5, 6, rel,
            [[rpro, r]], [np, tv(n, n)])
add: expect(6, np)
complete(4,5-4,6)
complete(3,5-3,6)
complete(2,5-2,6)
complete(1,5-1,6)
complete(0,5-0,6)
scan(7,8,v)
add: ctrees(7, 8, iv, [iv, v])
add: ctrees(7, 8, tv(n, n), [tv(n, n), v])
complete(6,6,7)
complete(5,6,7)
add: ptrees(5, 7, rel,
            [[rpro, r], [np, [n, n]]], [tv(n, n)])
add: expect(7, tv(n, n))
complete(5,5,7)
complete(4,6-4,7)
complete(3,6-3,7)
complete(2,6-2,7)
complete(1,6-1,7)
complete(0,6-0,7)
complete(7,7,8)
complete(6,7-6,8)
complete(5,7,8)
add: ctrees(5, 8, rel,
            [rel, [rpro, r], [np, [n, n]],
             [tv(n, n), v]])
complete(5,6-5,8)
complete(4,7-4,8)
```

```
complete(3,7-3,8)
complete(2,7-2,8)
complete(1,7-1,8)
complete(0,7-6,8)
complete(0,5,8)
add: ctrees(0, 8, np,
            [np, [det, d],
             [ap, [pp(p1), [pr(p1), p],
                   [np, [n, n]]], [a(p1), a]],
             [n, n],
             [rel, [rpro, r], [np, [n, n]],
              [tv(n, n), v]]]])
add: ctrees(0, 8, np,
            [np, [det, d],
             [ap, [pp(p2), [pr(p2), p],
                   [np, [n, n]]], [a(p2), a]],
             [n, n],
             [rel, [rpro, r], [np, [n, n]],
              [tv(n, n), v]]]])
complete(0,4-0,8)
```

Bem. Bei einer großen Grammatik (5.800 Regeln) war der Earley-Parser 3x so schnell wie der Leftcorner-Parser.