Verwendung von $<^*_{lc}$ im Parser

$$<^*_{lc} := \{(B, A) \mid \exists\gamma \in Cat^*\ A \Rightarrow^* B\gamma\}$$

Die atomare Kategorie `B` kann als linke Ecke einer zusammengesetzten Kategorie `A` auftreten, wenn sie ein Spezialfall `AI` von `A` ist oder zu einem solchen in der `first`-Relation steht:

⟨*leftcorner-rtc*⟩≡

```
leftcorner_rtc(B,A) :-
    instance(AI,A), (B = AI ; first(B,AI)).
```

Wird ein leeres Stück der Eingabe als Ausdruck der Kategorie `B` verwendet, so erlauben wir nur eine *kanonische* Analyse, `[B,[]]`, als Darstellung aller Ableitungen $B \Rightarrow^* \epsilon$.

⟨*canonical-trees*⟩≡

```
canonicalTrees([B|Bs],[[B,[]]|BsTrees]) :-
    canonicalTrees(Bs,BsTrees).
canonicalTrees([],[]).
```

Voraussage brauchbarer Regeln für die Left-Corner-Analyse

Ist E die für ein Präfix der Resteingabe „erwartete“ Kategorie, D die Kategorie eines nicht-leeren, schon erkannten Anfangsstücks der Resteingabe und b das auf dieses Anfangsstück folgende Eingabesymbol (bzw. seine Kategorie), so sei

$$\begin{aligned} &\mathit{predict}_{lc}(E, D, c) := \\ &\quad \{A \to \alpha B\beta_1 \cdot C\beta_2) \in \dot{P} \mid \\ &\qquad A <^*_{lc} E,\ \alpha\beta_1 \Rightarrow^* \epsilon,\ B \Rightarrow^* D,\ c <^*_{lc} C\} \end{aligned}$$

Diese Teilmenge von $\dot{P}$ kann für jedes (E, D, b) unabhängig von der Eingabe vorberechnet werden.

Der frühere Left-Corner-Erkenner kann damit beschleunigt werden, indem man das Probieren einer passenden Regel auf die jeweils als brauchbar vorausgesagten beschränkt:

Konfigurationen:	$(\eta, \gamma, w) \in \mathit{Cat}^* \times (\mathit{Cat} \cup \dot{P})^* \times \Sigma^*$
Anfang bzw. Ende:	$(S, \epsilon, a_1 \cdots a_n)$ bzw. (ϵ, S, ϵ)
Eingabe lesen:	$(\eta E, \gamma, aw) \vdash_a (\eta E, \gamma a, w)$, falls $a <^*_{lc} E$.
Partiell reduzieren:	$(\eta E, \gamma D, cw)$
	$\vdash (\eta EC, \gamma(A \to \alpha B\beta_1 \cdot C\beta_2), cw)$,
	falls $(A \to \alpha B\beta_1 \cdot C\beta_2) \in \mathit{predict}_{lc}(E, D, c)$
Vervollständigen 1:	$(\eta B, \gamma(A \to \alpha \cdot B\beta_1 C\beta_2)D, cw)$
	$\vdash (\eta C, \gamma(A \to \alpha B\beta_1 \cdot C\beta_2), cw)$
	falls $B \Rightarrow^* D$, $\beta_1 \Rightarrow^* \epsilon$, und $c <^*_{lc} C$,
Vervollständigen 2:	$(\eta B, \gamma(A \to \alpha \cdot B\beta)D, w) \vdash (\eta, \gamma A, w)$,
	falls $B \Rightarrow^* D$ und $\beta \Rightarrow^* \epsilon$

TABELLENVERFAHREN (CHART PARSER)

Bei den Analyseverfahren mit Rücksetzungen wird ein Teilausdruck bei einer Rücksetzung i.a. nochmals analysiert. Das vermeidet man bei den *Tabellenverfahren*, indem man die Analysen von Teilausdrücken in einer Tabelle speichert und bei Bedarf dort nachschlägt.

Braucht man nur ein Erkennungsverfahren, so speichert man in der Tabelle T zu jedem Teilausdruck v der Eingabe w alle Kategorien A, die v hat, d.h. $T(v) = \{\, A \mid A \Rightarrow^* v \,\}$.

1. Der Erkenner von Cocke, Younger, Kasami (CYK)

Vor.: Kontextfreie Grammatik $G = (N, \Sigma, S, P)$ in *Chomsky-Normalform*: alle Regeln haben die Form $(A \to a)$ oder $(A \to BC)$ mit $A, B, C \in N, a \in \Sigma$.

Eingabe: $w = a_1 \cdots a_n \in \Sigma^+$.

Methode: Konstruiere für die Positionen $0 \leq i < j \leq n$ alle *Kanten* $i \xrightarrow{A} j$ nach den Regeln:

$$\frac{a = a_{i+1}, \quad (A \to a) \in P}{i \xrightarrow{A} i+1} \quad (\mathit{Scan})$$

$$\frac{i \xrightarrow{B} k, \quad k \xrightarrow{C} j, \quad (A \to BC) \in P}{i \xrightarrow{A} j} \quad (\mathit{Complete})$$

Ausgabe: $0 \xrightarrow{S} n$ wird konstruiert $\iff$ $w \in L(G)$.

Das folgt aus: $i \xrightarrow{A} j$ wird konstruiert $\iff A \Rightarrow^* a_{i+1} \cdots a_j$.

Da es nur endlich viele Kanten $i \xrightarrow{A} j$ zu w gibt, terminiert das Verfahren (bei geeigneter Verwaltung).

Das CYK-Verfahren ist ein „bottom-up"-Verfahren: die Konstruktion der Kanten beginnt bei den $i \xrightarrow{a_{i+1}} i+1$ und nutzt S nicht aus.

Programmschleife zur Konstruktion der Kanten

Die Kanten stellt man als Eintrag einer $n \times n$-Tabelle T dar:

$$T(i,j) = \{\, A \mid i \xrightarrow{A} j \,\} = \{\, A \mid A \Rightarrow^* a_{i+1} \cdots a_j \,\}.$$

Man kann die Kanten in verschiedener Reihenfolge bilden, z.B. zuerst alle Kanten $i \xrightarrow{A} k$ mit $i < k < j$, dann alle $j-1 \xrightarrow{A} j$, dann alle $j-2 \xrightarrow{A} j$, bis man alle mit $i < k \leq j$ hat, usw.:

for $j = 1$ to n do
  $T(j-1, j) := \{\, A \in N \mid (A \to a_j) \in P \,\}$;
  if $j > 1$ then
    for $i = j-2$ to 0 do
      $T(i,j) := \bigcup_{i<k<j} \{\, A \in N \mid B \in T(i,k),\, C \in T(k,j),$
                  $(A \to BC) \in P \,\}$
      od
  od

Aufgabe: Konstruiere die Kanten in der Reihenfolge: zuerst alle Kanten der Länge 1, die $i \xrightarrow{A} i+1$, dann die der Länge 2, die $i \xrightarrow{A} i+2$, usw.

Beispiel

Komplexität der Erkennung mit CYK

- Für $i = 0, \ldots, n-1$, initialisiere Feld $T(i, i+1)$. Dazu muß für jede Regel geprüft werden, ob sie von der Form $A \to a_i$ ist. Das erfordert jeweils $O(|P|)$ Schritte.

  Insgesamt also $O(|w| * |P|) \leq O(|w| * |G|)$ Schritte.

- Angenommen, $T(i, j)$ ist für alle Teilwörter $a_{i+1} \ldots a_j$ der Länge $j - i < k$ (endgültig) bestimmt.

  Zum Auffüllen der $T(i', j')$ mit $|j' - i'| = k$ ist für alle Regeln $A \to BC$ und alle k' mit $i' < k' < j'$ zu prüfen, ob $B \in T(i', k')$ und $C \in T(k', j')$.

  Um $T(i', j')$ zu füllen, braucht man

  $$O(|P| \times (j' - i') \times |N|)$$

  viele Schritte (auf einer RAM), wobei höchstens $|N|$ Schritte für die Suche durch das Feld $T(i, j)$ nötig sind.

- Um alle Felder von T zu füllen, braucht man also insgesamt höchstens

$$\begin{aligned} & O(|w| * |G|) + \sum_{i<j\leq|w|} O(|P| \times (j-i) \times |N|) \\ = \quad & O(|w| * |G|) + O(|P| \times |N| \times \sum_{j\leq|w|} \frac{j(j+1)}{2}) \\ \leq \quad & O(|P| \times |N| \times \frac{|w|^3}{3}) = O(|w|^3 \times |G|^2) \end{aligned}$$

Für Regeln der Form $(A \to B_1 \cdots B_k)$ ist der Aufwand größer!

2. Der Erkenner von Earley

Das Verfahren von Earley kann man als eine Verbesserung des CYK-Verfahrens ansehen: die Grammatik braucht nicht in Chomsky-Normalform zu sein, und die Analyse benutzt die Kategorie S als „Top-Down-Filter", d.h. für die Eingabe werden nur ihre Analysen als S-Ausdruck konstruiert.

Vor.: Beliebige kontextfreie Grammatik $G = (N, \Sigma, S, P)$

Eingabe: $w = a_1 \cdots a_n \in \Sigma^*$

Tethode: Konstruiere für die Positionen $0 \leq i \leq j \leq n$ alle Kanten $i \stackrel{A \to \alpha \cdot \beta}{\longrightarrow} j$, wo $(A \to \alpha\beta) \in P$, nach den Regeln:

$$\frac{(S \to \alpha) \in P}{0 \stackrel{S \to \cdot \alpha}{\longrightarrow} 0} \quad (\mathit{Predict})$$

$$\frac{i \stackrel{A \to \alpha \cdot C\beta}{\longrightarrow} j, \quad (C \to \gamma) \in P}{j \stackrel{C \to \cdot \gamma}{\longrightarrow} j} \quad (\mathit{Predict})$$

$$\frac{i \stackrel{A \to \alpha \cdot a\beta}{\longrightarrow} j, \quad a = a_{j+1}}{i \stackrel{A \to \alpha a \cdot \beta}{\longrightarrow} j+1} \quad (\mathit{Scan})$$

$$\frac{i \stackrel{A \to \alpha \cdot C\beta}{\longrightarrow} k, \qquad k \stackrel{C \to \gamma \cdot}{\longrightarrow} j}{i \stackrel{A \to \alpha C \cdot \beta}{\longrightarrow} j} \quad (\mathit{Complete})$$

Ausgabe: Eine Kante der Form $0 \stackrel{S \to \alpha \cdot}{\longrightarrow} n$ wird erzeugt

$$\iff \quad w \in L(G).$$

Zum Beweis zeigt man:

Beh.: Die Kante $i \stackrel{A \to \alpha \cdot \beta}{\longrightarrow} j$ wird genau dann konstruiert, wenn

(i) $S \Rightarrow^* a_1 \cdots a_i A \gamma$ für ein $\gamma \in (N \cup \Sigma)^*$, und

(ii) $\alpha \Rightarrow^* a_{i+1} \cdots a_j$.

Bew. $\Longrightarrow$ (Korrektheit der Regeln): Zeige (i), (ii) durch Induktion über die Anzahl der Schritte zur Erzeugung der Kante.

$\Longleftarrow$ (Vollständigkeit der Regeln): Für $(A \to \alpha\beta) \in P$ gelte (i) und (ii). Dann gibt es Linksableitungen

$$S \Rightarrow^{m_1} a_1 \cdots a_i A \gamma \quad \text{und} \quad \alpha \Rightarrow^{m_2} a_{i+1} \cdots a_j$$

bestimmter Längen m_1, m_2. Durch Induktion über $m_1 + m_2$ zeigt man, daß $i \stackrel{A \to \alpha \cdot \beta}{\longrightarrow} j$ konstruiert wird.[3]

Die Analyse von Earley ist eine Mischung aus Top-Down- und Bottom-Up-Vorgehen:

- (Predict) beginnt zu Erwartungskategorien C „top-down“ neue partielle Erkennungsversuche auf. (Divergiert bei $(C \to C\gamma')$, wenn man die *Analysen* aufbaut!)
- (Complete) erweitert „bottom-up“ eine partielle erkannte A-Phrase durch eine vollständig erkannte C-Phrase.

[3]Man braucht die Hilfsbehauptung, daß für $m_1 > 0$ ein $S \Rightarrow^{m_1 - 1} a_1 \cdots a_k A' \gamma'$ entsteht und alle $k \stackrel{A' \to \cdot \alpha'}{\longrightarrow} k$ erzeugt werden.

Earley-Erkenner in Prolog

⟨*Kategorien- und Regelformat*⟩≡

```
Cat ::= Atom
Rule ::= (Cat --> [Cat1,...,Catn]).
```

⟨*earley-doerre.pl*⟩≡

```
:- dynamic kante/5.

erkenne(_,_) :-              % Predict(0)
        retractall(kante(_I,_J,_A,_alpha,_beta)),
        start_symbol(S),
        (S --> Alpha),
        closure(0,0,S,[],Alpha),
        fail.
erkenne(Wortfolge,Kategorie) :-
        erkenne(0,Wortfolge,Kategorie).

erkenne(I,[Wort|Rest],Kategorie) :-
        J is I + 1,
        shift_e(I,J,Wort),
        erkenne(J,Rest,Kategorie).
erkenne(I,[],S) :-           % Eingabe erkannt?
        start_symbol(S),
        kante(0,I,S,_Alpha,[]).

shift_e(I,J,Wort) :-         % Scan(a_(i+1))
        lexword(I,Wort,Kategorie),
        closure(I,J,Kategorie,[Wort],[]),
        fail.
shift_e(_,_,_).
```

Earley-Erkenner in Prolog (Forts.)

Entsteht eine neue Kante $i \xrightarrow{B \to \beta\cdot} j$, so werden alle „wartenden“ Kanten $k \xrightarrow{A \to \alpha\cdot B\beta} i$ damit verbunden (Complete):

$\langle$*earley-doerre.pl*$\rangle$+≡

```
closure(I,J,Kopf,Geschlossen,[]) :-
        neue_kante(I,J,Kopf,Geschlossen,[]),
        complete(I,J,Kopf).
complete(I,J,B) :-
        kante(K,I,A,Alpha,[B|Beta]),
        append(Alpha,[B],AlphaB),
        closure(K,J,A,AlphaB,Beta),
        fail.
complete(_,_,_).
```

Löschbare Kategorien B in $i \xrightarrow{A \to \alpha\cdot B\beta} j$ können (Backtracking!) übersprungen werden:

$\langle$*earley-doerre.dcg*$\rangle$≡

```
closure(I,J,Kopf,Geschlossen,[B|Bs]) :-
        deletable(B),
        append(Geschlossen,[B],Geschlossen1),
        closure(I,J,Kopf,Geschlossen1,Bs).
```

Earley-Erkenner in Prolog (Forts.)

Ist die Kante neu, so muß (Predict) angewendet werden:

⟨*earley-doerre.pl*⟩+≡

```
closure(I,J,Kopf,Geschlossen,[B|Bs]) :-
        neue_kante(I,J,Kopf,Geschlossen,[B|Bs]),
        expand_e(J,B).

expand_e(J,A) :-                 % Predict(J)
        (A --> Alpha),
        closure(J,J,A,[],Alpha),
        fail.
expand_e(_,_).
```

Nur *neue* Kanten, die keine Instanz einer vorhandenen Kante sind, sollen in die Tabelle eingetragen werden:

⟨*earley-doerre.pl*⟩+≡

```
neue_kante(I,J,Kopf,Geschl,Offen) :-
        known_instance(kante(I,J,Kopf,Geschl,Offen)),
        !,fail.
neue_kante(I,J,Kopf,Geschlossen,Offen) :-
        assertz(kante(I,J,Kopf,Geschlossen,Offen)).

known_instance(Term) :-
        \+ \+ (numbervars(Term,const,0,_), Term).
```

Durch `numbervars(Term,const,0,N)` von Prolog werden die Variable in `Term` durch Konstante `const(0)` usw. ersetzt; danach rufen wir `Term` auf, um zu testen, ob dies beweisbar (also Instanz einer Kante) ist, und durch doppelte Negation `\+` wird die Einsetzung in `Term` rückgängig gemacht.