eg: BUILDING a STRONG LL(1) GRAMMAR

- $\Rightarrow G_{AE}$:
$$\begin{aligned} S' \rightarrow &\quad S\$ \\ S \rightarrow &\quad A \\ A \rightarrow &\quad T \mid A + T \\ T \rightarrow &\quad b \mid (A) \end{aligned}$$

- A is left-recursive: re-write

$$\begin{aligned} A \rightarrow &\quad TA' \\ A' \rightarrow &\quad +TA' \mid \epsilon \end{aligned}$$

- FIRST sets:

FIRST$_1(S) = \{b, (\}$ $\Rightarrow$ FIRST$_1(A) = \{b, (\}$

FIRST$_1(A') = \{+, \epsilon\} \Rightarrow$ FIRST$_1(T) = \{b, (\}$

- FOLLOW sets:

FOLLOW$_1(S) = \{\$\} \Rightarrow$ FOLLOW$_1(A) = \{\$,)\}$

FOLLOW$_1(A') = \{\$,)\}$

FOLLOW$_1(T) = \{\$,), +\}$=FIRST$_1(A')$FOLLOW$_1(A')$

- LA sets:

LA$_1(S \rightarrow A) = \{b, (\}$

LA$_1(A \rightarrow TA') = \{b, (\}$

LA$_1(T \rightarrow b) = \{b\}$

$$\mathsf{LA}_1(T \rightarrow (A)) = \{(\}$$

$$\mathsf{LA}_1(A' \rightarrow \epsilon) = \{\$,)\}$$

$$\mathsf{LA}_1(A' \rightarrow +TA') = \{+\}$$

- A strong LL(k) parser

```
input: LL(k) grammar;
       input string p;
       LAk sets

1. Start with S (q=S)
2. repeat
     let q=uAv   (A is leftmost variable;
         p=uyz    u= prefix of q;
                  y is the lookahead string;
                  length(y)=k)

     if y is in LAk set of a rule A->w
         q=uAv => uwv  can de done deterministically
         let q=uwv
   until q=p or y is not in any LAk set
3. if q=p then accept, otherwise reject
```

exercise: try input (b+b) on LL(1) version of G_{AE}

- A no-lookahead algorithm would backtrack 3 times for the same input.

- LL(k) versus strong LL(k) grammars:

  A grammar is LL(k) but not strong LL(k) if the LA sets of rules are not necessarily disjoint but the LA sets of any sentential form is unique.

$$
\begin{aligned}
S \rightarrow &\quad Aabd \mid cAbcd \\
A \rightarrow &\quad a \mid b \mid \epsilon
\end{aligned}
$$

  $\text{LA}_2(A \rightarrow a) = \{aa, ab\}$

  $\text{LA}_2(A \rightarrow \epsilon) = \{ab, bc\} \Rightarrow$ not strong LL(2)

- $\text{LA}_k(uAv, A \rightarrow w) = \text{FIRST}_k(wv)$

$\text{LA}_2(Aabd, A \rightarrow a) = \{aa\}$

$\text{LA}_2(Aabd, A \rightarrow b) = \{ba\}$

$\text{LA}_2(Aabd, A \rightarrow \epsilon) = \{ab\}$

Similarly LA_2 sets of $cAbcd$ also form a partition. $\Rightarrow$ LL(2)

- In general though, number of sentential forms in a grammar is quite large (sometimes can only be described by a pattern).

- LA sets must be calculated on-the-fly $\Rightarrow$ costly

- TABLE-DRIVEN LL. rows: variables. columns: LA_k sets

  if the table has multiple entries, it is not LL(k)

- Constructing the LL(k) table (slightly different than the algorithm in p.190; this uses LA sets directly)

```
input:  G=(Alphabet,V,P,S)
        LAk sets for all rules in P
 output: parsing table M

 1. For all A->w in P

     2. M[A,u] contains A->w ; for each terminal
                               string u in LAk(A->w)

     3. M[A,$] contains A->w if $ is in LAk(A->w)
```

4. Make all empty entries of M error states

	b	(	)	+	$
S′	S′->S$	S′->S$			
S	S->A	S->A			
A	A->TA′	A->TA′			
A′			A′-> null	A′->+TA′	A′->null
T	T->b	T->(A)			

parsing: start with S’, find the right rule from LA sets. if next symbol is $, see if $q = p$ after last rule.