WARSAW UNIVERSITY OF TECHNOLOGY
DEVELOPMENT PROGRAMME

Compiling Techniques - ECOTE

part 5- Recursive Descent parsers

DSc dr Ilona Bluemke

HUMAN CAPITAL
HUMAN – BEST INVESTMENT!

EUROPEAN UNION
EUROPEAN
SOCIAL FUND

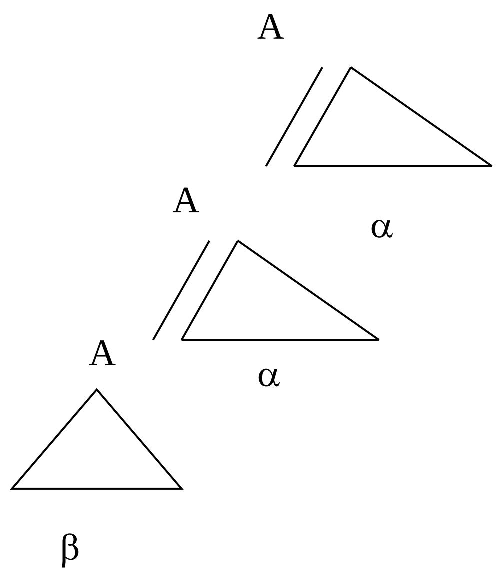
Project is co-financed by European Union within European Social Fund

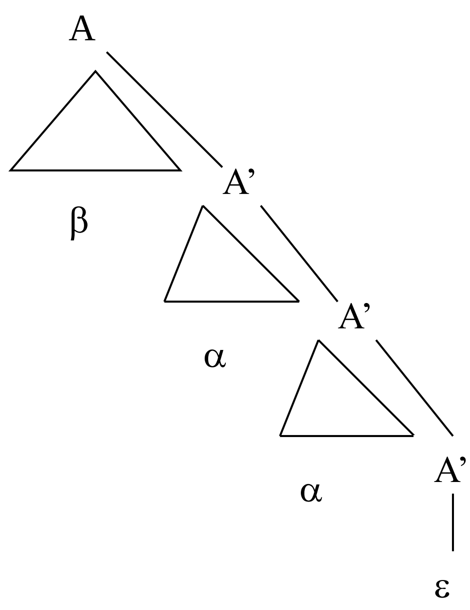
Elimination of left recursion

Left-recursive pair of productions:

$A \rightarrow A\alpha \mid \beta$ where β does not begin with A

Can be replaced by: $A \rightarrow \beta A'$ $A' \rightarrow \alpha A' \mid \varepsilon$





In general to eliminate left-recursive among all A-productions we group them as:

$A \to A\alpha_1 \mid A\alpha_2 \mid \ldots \mid A\alpha_m \mid \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$

where β_i does not begin with A

and replace by:

$\boldsymbol{A \to \beta_1 A' \mid \beta_2 A' \mid \ldots \mid \beta_n A'}$

$\boldsymbol{A' \to \alpha_1 A' \mid \alpha_2 A' \mid \ldots \mid \alpha_m A' \mid \varepsilon}$

This process will eliminate immediate left-recursion but will not eliminate left-recursion in 2 or more steps:

$S \to Aa \mid b$

$A \to Ac \mid Sd \mid e$

S is left-recursive :$S \Rightarrow Aa \Rightarrow Sda$

Algorithm to eliminate left-recursion from grammar

Grammar without cycles ($A \Rightarrow^{+} A$), ε - productions, resulting grammar may have ε - productions.

1. Arrange nonterminals of G in some order $A_1, A_2, \ldots, A_n$
2. **for** i:= 1 **to** n **do**

   begin for j:= 1 **to** i-1 **do**

replace each production of the form $A_i \rightarrow A_j\gamma$

by the productions :

$A_i \rightarrow \delta_1\gamma \mid \delta_2\gamma \mid \ldots \mid \delta_k\gamma$ where

$A_j \rightarrow \delta_1 \mid \delta_2 \mid \ldots \mid \delta_k$;

eliminate immediate recursion among A_i -productions

   end

Exercise

Consider grammar:

1. $S \rightarrow a$
2. $S \rightarrow \wedge$
3. $S \rightarrow (T)$
4. $T \rightarrow T,S$
5. $T \rightarrow S$

Eliminate left recursion from this grammar

Elimination of left recursion

- Assume that S < T
- Consider **S** productions
  - (no recursion)
- Consider **T** productions: α = ,S β= S

T $\rightarrow$ T,S and T $\rightarrow$ S transformed into :

T $\rightarrow$ S T' | S and T' $\rightarrow$,S T' | ε

$S \rightarrow a \mid \wedge \mid (T)$

$T \rightarrow T,S \mid S$

Transformed into:

$S \rightarrow a \mid \wedge \mid (T)$

$T \rightarrow S\ T' \mid S$

$T' \rightarrow ,S\ T' \mid \varepsilon$

WARSAW UNIVERSITY OF TECHNOLOGY
DEVELOPMENT PROGRAMME

Example 1

G_0 = <{a, +, *, (,)}, {A, B, S},*P* ,S>

P = {S→S+A | A, A→A*B | B, B→(S) | a}

- B < A < S ordering of nonterminals
  - B→(S) | a
  - A→A*B | (S) | a substitution of B

  Left recursion : A→(S) | a | (S) A' | a A'

  A' → * B | * B A'
  - S→S+A | (S) | a | (S) A' | a A' (substitution of A)

Example 2

- $S \rightarrow S+A \mid (S) \mid a \mid (S)\ A' \mid a\ A'$ (substitution of A)

Left recursion :

$S \rightarrow (S) \mid a \mid (S)\ A' \mid a\ A'$

$S \rightarrow (S)\ S' \mid a\ S' \mid (S)\ A'\ S' \mid a\ A'\ S'$

$S' \rightarrow +A\ S' \mid +A$

Example 3

The same grammar but different ordering of symbols: $S < A < B$

- $S \to S+A \mid A$ => $S \to AS' \mid A \quad S' \to +A \mid +AS'$
  - Reduction $S' \to +S$
  - S' substituted $S \to A + S \mid A$
- $A \to A*B \mid B$ => $A \to BA' \mid B \quad A' \to *B \mid *BA'$
  - Reduction $A' \to * A$
  - A' substituted $A \to B * A \mid B$
- $B \to (S) \mid a$

Recursive-descent parsing

Top-down parser, no backtracking

Given the current input symbol ***a***

and the nonterminal A

which of the alternates of production

$$A \rightarrow \alpha_1 \mid \alpha_2 \mid \ \dots \ \mid \alpha_m$$

is the unique alternates

that derives a string beginning with ***a***.

Example

$A \rightarrow$ **if** cond **then** A **else** A |

while cond **do** A |

begin A-list **end**

the keywords **if while begin** choose the alternate

ε - production

if one alternate for A is $A \rightarrow \varepsilon$, and none of the alternates derives a string beginning with ***a*****,** then on input ***a*** we may expand A with

$A \rightarrow \varepsilon$

R-D parser uses a set of **recursive procedures** (one for each nonterminal) to recognise its input without backtracking.

Example

Grammar **without left-recursion**

$S \rightarrow AS'$

$S' \rightarrow + AS' \mid \varepsilon$

$A \rightarrow BA'$

$A' \rightarrow * BA' \mid \varepsilon$

$B \rightarrow (S) \mid a$

Procedures:

```
void S()
{     A();
   Sprime()
}


void A()
{     B();
   Aprime()
}
```

```
void Sprime()
{     if(symbol=='+')
   then
   {
   advance ()
   A();
   Sprime()
   }
}
```

```
void Aprime()
{ if (symbol == '*') then
  {  advance ()
     B();
     Aprime() }
}
```

Procedure B

```
 void B()
{ if (symbol == ‘a’) then  advance ()
else if (symbol == ‘(’) then
   {    advance();
        S();
   if (symbol == ‘)’) then
        advance(); else error();}
}
```

Left factoring

A $\rightarrow$ **if** cond **then** A **else** A $|$ **if** cond **then** A

On seeing symbol **if** it is not possible to tell which production to choose.

Left factoring –process of factoring out the common prefixes of alternates

1. **$A \rightarrow \alpha \beta \mid \alpha\gamma$**

Becomes:

$$A \rightarrow \alpha A'$$
$$A' \rightarrow \beta \mid \gamma$$

2. (**if-then-else**)

$S \rightarrow i\ C\ t\ S \mid i\ C\ t\ S\ e\ S \mid a$

$C \rightarrow b$

Becomes:

$$S \rightarrow i\ C\ t\ S\ S' \mid a$$
$$S' \rightarrow e\ S \mid \varepsilon$$
$$C \rightarrow b$$

Transition diagrams for R-D parser

- plan, flowchart for lexical analyser
- plan for recursive-descent parser

- one transition diagram for each nonterminal
- labels – tokens or nonterminals

token (terminal) – take this transition if that token is the next input symbol

nonterminal A– the transition diagram for **A** should be called

For each **nonterminal** do:

1. create an initial and final (return) state
2. for each production $A \rightarrow X_1\ X_2\ \ldots\ X_n$ create a path from the initial to the final state, with **edges labelled** $X_1, X_2, \ldots X_n$

transition diagrams:

For grammar

$S \rightarrow AS'$

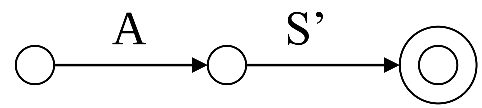
$S' \rightarrow +\ AS' \mid \varepsilon$

$A \rightarrow BA'$

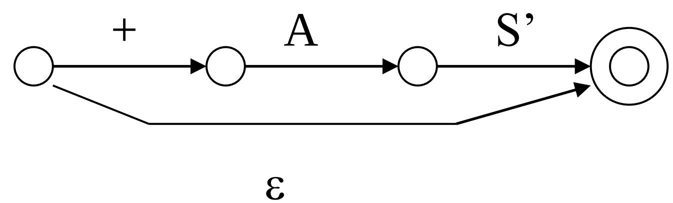
$A' \rightarrow *\ BA' \mid \varepsilon$

$B \rightarrow (\ S\ ) \mid a$

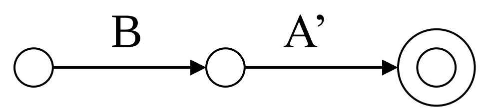
S:



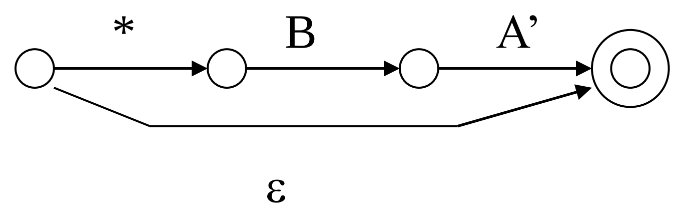
S':



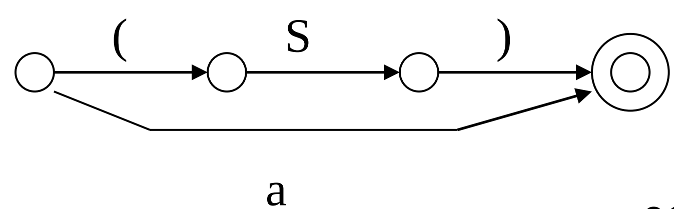
A:



A':

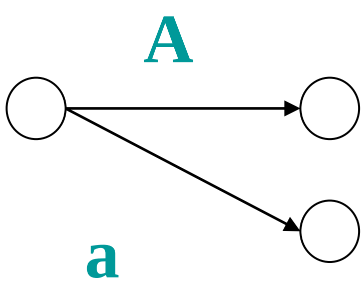


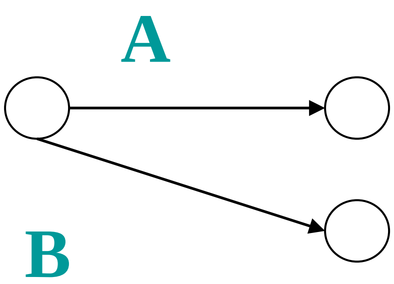
B:



Transition diagrams should be deterministic – subset construction algorithm does not work (cannot remember how many recursive calls are made).

Problems:





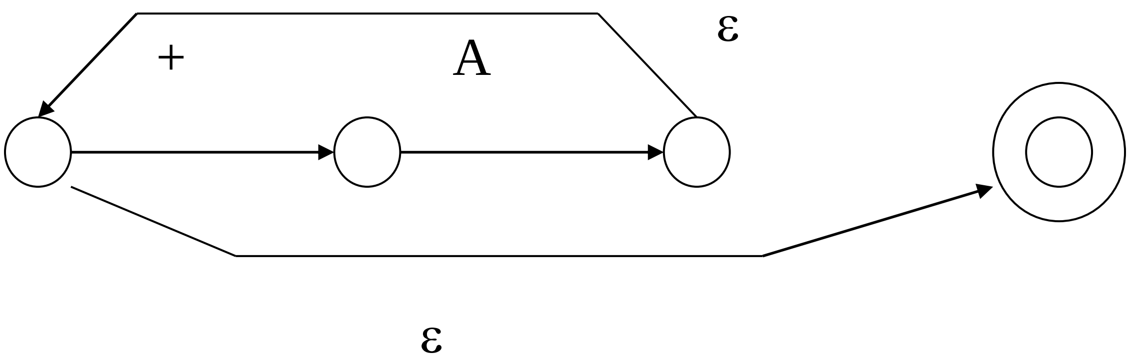
First symbol derived from **A** should not be **a**

First symbols derived from A and B should be different

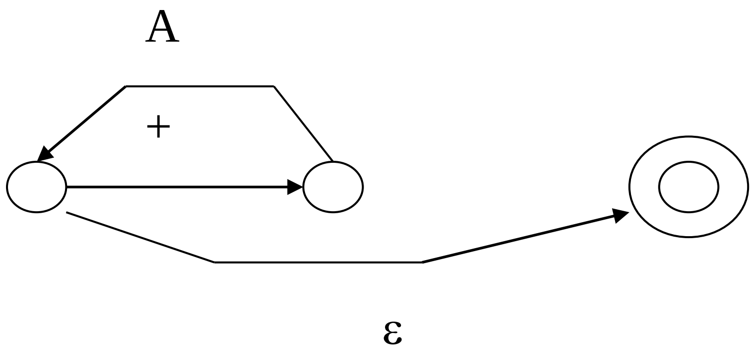
Simplifying transition diagrams (substituting diagram in one another)

The call of S' on itself can be replaced by a jump to the initial state of S' diagram:

S':

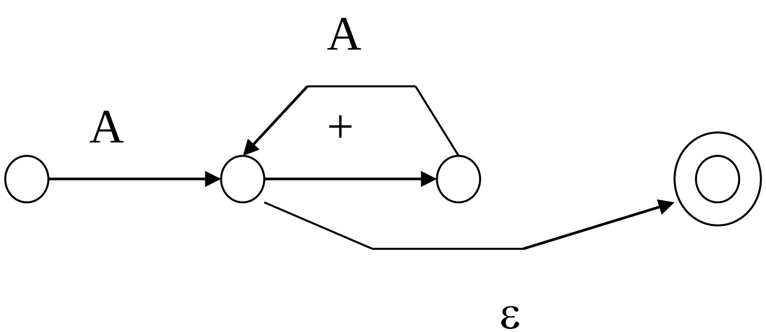


Equivalent diagram for A’:

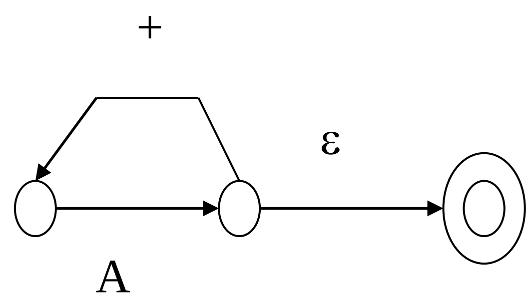


For S:

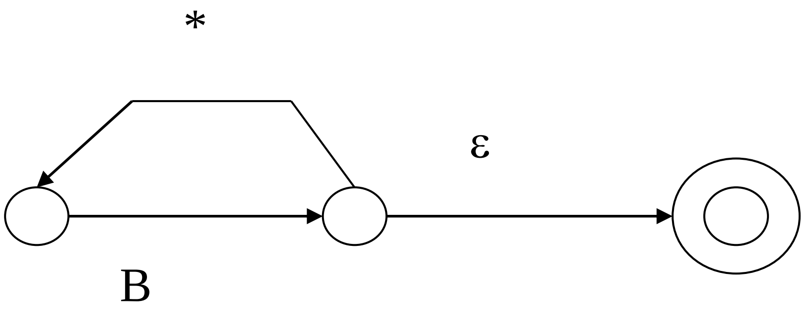
a)



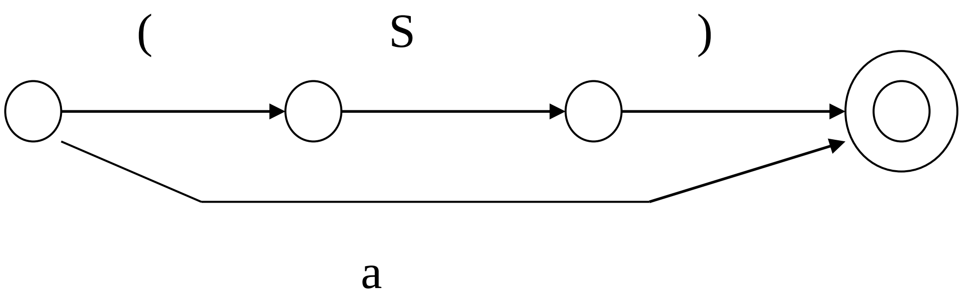
b)



A



B



Code example for A:

```
void A()
{
    B();
  while (symbol == '*')
    {
    advance ();
    B();
    }
}
```

First and Follow

If α is any string of grammar symbols, let **First(α)** be the set of terminals that begin strings derived from α

$\forall\ (a \in T)\quad First_k\ (a) = \{a\}$

$\forall\ (A \in N)\quad First_k(A) = \{x \in T^{*k}\!:\ (A \Rightarrow^* x \wedge |x| <= k \vee A \Rightarrow^* xy \wedge |x| = k\ \}$

$First_k\ (\varepsilon) = \{\varepsilon\}$

$First_k(\alpha_1\alpha_2 \ldots \alpha_n) = First_k(\alpha_1) \bullet_k First_k(\alpha_2) \bullet_k \ldots First_k(\alpha_n)$

where $\bullet_k$ is a concatenation of the length k.

To compute **$First_1(\alpha)$**

for all grammar symbols α

apply the following rules until no more terminals or ε can be added to any First set:

1. if α is a **terminal** then $First_1(\alpha)$ is $\{\alpha\}$
2. if α is **nonterminal** and if $\alpha \rightarrow a\beta$ is a production then add a to $First_1(\alpha)$, if $\alpha \rightarrow \varepsilon$ is a production then add ε to $First_1(\alpha)$.

$\mathbf{First_1}$

3. if $\alpha \to \beta_1\ \beta_2\ \dots\ \beta_n$ is a production, then

   for all ***i*** such that all of $\beta_1\ \beta_2\ \dots\ \beta_{i-1}$ are nonterminals and $First_1(\beta_j)$, contains ε

   for $j = 1, 2, \dots, i-1$ (i.e. $\beta_1\ \beta_2\ \dots\ \beta_{i-1} \Rightarrow^* \varepsilon$),

   add every non- ε symbol in $First_1(\beta_i)$ to $First_1(\alpha)$.

   If $\varepsilon \in$ **$First_1(\beta_j)$** for all $j = 1, 2, \dots, n$, then add **ε to $First_1(\alpha)$.**

Example

$S \rightarrow AS'$

$S' \rightarrow +AS' \mid \varepsilon$

$A \rightarrow BA'$

$A' \rightarrow *BA' \mid \varepsilon$

$B \rightarrow (S) \mid a$

$First_1(B) = \{(, a\}$

$First_1(A') = \{*, \varepsilon\}$

$First_1(S') = \{+, \varepsilon\}$

$First_1(S) = First_1(A) = \{(, a\}$

Example 2

$S \rightarrow a \mid \wedge \mid (T)$

$T \rightarrow S\ T' \mid S$

$T' \rightarrow ,S\ T' \mid \varepsilon$

$First_1(S) = \{a, (, \wedge\}$

$First_1(T) = First_1(S) = \{a, ,(, \wedge\}$

$First_1(T') = \{, \varepsilon\}$

$\mathbf{Follow_1\ (A)}$

The set of terminals (a) that can appear immediately to the right of A in some sentential form,

that is, $S \Rightarrow^* \alpha A a \beta$

for some α and β.

If **A** can be the rightmost symbol in some sentential form, then $\varepsilon \in \mathbf{Follow_1\ (A)}$.

To compute **$Follow_1(A)$** for all nonterminals A, apply the following rules until nothing can be added to any Follow set:

1. ε is **$Follow_1(S)$**, where S is the start symbol
2. if there is a production **$A \rightarrow \alpha B \beta$**,

then **everything** in **$First_1(\beta)$**

but ε, is in $Follow_1(B)$.

Note that ε may still wind up in $Follow_1(B)$ by rule 3.

$Follow_1(A)$

3. if there is a production $A \rightarrow \alpha B$, or a production $A \rightarrow \alpha B \beta$ where

$First_1(\beta)$ contains ε (i.e. $\beta \Rightarrow^* \varepsilon$), then everything in

$Follow_1(A)$ is in $Follow_1(B)$.

Example

$S \rightarrow AS'$

$S' \rightarrow + AS' \mid \varepsilon \quad A \rightarrow BA'$

$A' \rightarrow * BA' \mid \varepsilon \quad B \rightarrow (S) \mid a$

$First_1(B) = \{(, a\}$ $\quad$ $First_1(A') = \{*, \varepsilon\}$

$First_1(S') = \{+, \varepsilon\}$ $\quad$ $First_1(S) = First_1(A) = \{(, a\}$

$Follow_1(S) = Follow_1(S') = \{), \varepsilon\}$

$Follow_1(A) = Follow_1(A') = \{+,), \varepsilon\}$

$Follow_1(B) = \{+, *,), \varepsilon\}$

Example 2

$S \rightarrow a \mid \wedge \mid (T)$

$T \rightarrow S\ T' \mid S$

$T' \rightarrow ,S\ T' \mid \varepsilon$

$First_1(S) = \{a, (, \wedge\}$

$First_1(T) = First_1(S) = \{a, ,(, \wedge\}$

$First_1(T') = \{, \varepsilon\}$

$Follow_1(S) = \{\varepsilon,$ „ , ”,) }

$Follow_1(T) = \{\ )\ \}$

$Follow_1(T') = \{)\ \}$

WARSAW UNIVERSITY OF TECHNOLOGY
DEVELOPMENT PROGRAMME

Compiling Techniques - ECOTE

Recursive Descent parsers

end of part 5

HUMAN CAPITAL
HUMAN – BEST INVESTMENT!

EUROPEAN UNION
EUROPEAN
SOCIAL FUND

Project is co-financed by European Union within European Social Fund