Parsing Descendente (Parte 01)

Sumário

Parsing Universal × Descendente × Ascendente

Um analisador sintático (ou *parser*) é responsável por agrupar *tokens* em sentenças gramaticalmente corretas ou identificar erros quando isso não for possível.

Existem 3 tipos gerais de analisadores sintáticos:

- Universal
- Descendente
- Ascendeste

Parsing Universal × Descendente × Ascendente

Um analisador sintático (ou *parser*) é responsável por agrupar *tokens* em sentenças gramaticalmente corretas ou identificar erros quando isso não for possível.

Existem 3 tipos gerais de analisadores sintáticos:

- Universal
- Descendente
- Ascendeste

Parsing Universal × Descendente × Ascendente

Os métodos universais podem tratar quaisquer tipos de gramáticas (por ex.: CYK, Earley), porém eles são muito ineficientes para uso geral.

Os métodos descendentes (ou "*top-down*") constroem árvores de derivação do topo (raiz) para o fundo (folhas), enquanto os métodos ascendentes (ou "*bottom-up*") começam nas folhas e trabalham em direção à raiz.

Em ambos os casos, a entrada é geralmente varrida da esquerda para direita, um símbolo (ou *token*) por vez.

Parsing Universal × Descendente × Ascendente

Os métodos universais podem tratar quaisquer tipos de gramáticas (por ex.: CYK, Earley), porém eles são muito ineficientes para uso geral.

Os métodos descendentes (ou "*top-down*") constroem árvores de derivação do topo (raiz) para o fundo (folhas), enquanto os métodos ascendentes (ou "*bottom-up*") começam nas folhas e trabalham em direção à raiz.

Em ambos os casos, a entrada é geralmente varrida da esquerda para direita, um símbolo (ou *token*) por vez.

Parsing Universal × Descendente × Ascendente

Os métodos universais podem tratar quaisquer tipos de gramáticas (por ex.: CYK, Earley), porém eles são muito ineficientes para uso geral.

Os métodos descendentes (ou "*top-down*") constroem árvores de derivação do topo (raiz) para o fundo (folhas), enquanto os métodos ascendentes (ou "*bottom-up*") começam nas folhas e trabalham em direção à raiz.

Em ambos os casos, a entrada é geralmente varrida da esquerda para direita, um símbolo (ou *token*) por vez.

Parsing Descendente × Ascendente

Exemplo de Parsing Descendente × Ascendente

Considere GLC G: $S \to a\ A$
$A \to B\ c$
$B \to b$

A sentença abc pode ser obtida a partir da raiz da seguinte forma:

$$S \Rightarrow aA \Rightarrow aBc \Rightarrow abc$$

Também é possível se obter a mesma sentença a partir das folhas:

$$abc \mapsto aBc \mapsto aA \mapsto S$$

em que $\mapsto$ representa a substituição do lado direito de uma regra pelo não-terminal do lado esquerdo (ou "derivação reversa").

Parsing Descendente × Ascendente

Exemplo de Parsing Descendente × Ascendente

Considere GLC G: $S \to a\ A$
$A \to B\ c$
$B \to b$

A sentença abc pode ser obtida a partir da raiz da seguinte forma:

$$S \Rightarrow aA \Rightarrow aBc \Rightarrow abc$$

Também é possível se obter a mesma sentença a partir das folhas:

$$abc \mapsto aBc \mapsto aA \mapsto S$$

em que $\mapsto$ representa a substituição do lado direito de uma regra pelo não-terminal do lado esquerdo (ou "derivação reversa").

Parsing Descendente × Ascendente

Exemplo de Parsing Descendente × Ascendente

Considere GLC G: $S \to a\ A$
$A \to B\ c$
$B \to b$

A sentença abc pode ser obtida a partir da raiz da seguinte forma:

$$S \Rightarrow aA \Rightarrow aBc \Rightarrow abc$$

Também é possível se obter a mesma sentença a partir das folhas:

$$abc \mapsto aBc \mapsto aA \mapsto S$$

em que $\mapsto$ representa a substituição do lado direito de uma regra pelo não-terminal do lado esquerdo (ou "derivação reversa").

Parsing Recursivo Descendente

Objetivo

Determinar se uma sentença pode ser derivada a partir do símbolo da partida de uma gramática.

Abordagem

Substituir recursivamente um não-terminal pelo lado direita de uma produção.

A cada passo, deve-se manter controle sobre:

- Qual é o nó da árvore de derivação se está expandindo?
- Qual é a próxima *token* da sentença de entrada (***lookahead***)?
  - Isto ajuda a selecionar qual a regra que deve ser usada na substituição do não-terminal.

Parsing Recursivo Descendente

Objetivo

Determinar se uma sentença pode ser derivada a partir do símbolo da partida de uma gramática.

Abordagem

Substituir recursivamente um não-terminal pelo lado direita de uma produção.

A cada passo, deve-se manter controle sobre:

- Qual é o nó da árvore de derivação se está expandindo?
- Qual é a próxima *token* da sentença de entrada (***lookahead***)?
  - Isto ajuda a selecionar qual a regra que deve ser usada na substituição do não-terminal.

Parsing Recursivo Descendente

Objetivo

Determinar se uma sentença pode ser derivada a partir do símbolo da partida de uma gramática.

Abordagem

Substituir recursivamente um não-terminal pelo lado direita de uma produção.

A cada passo, deve-se manter controle sobre:

- Qual é o nó da árvore de derivação se está expandindo?
- Qual é a próxima *token* da sentença de entrada (***lookahead***)?
  - Isto ajuda a selecionar qual a regra que deve ser usada na substituição do não-terminal.

Parsing Recursivo Descendente

Objetivo

Determinar se uma sentença pode ser derivada a partir do símbolo da partida de uma gramática.

Abordagem

Substituir recursivamente um não-terminal pelo lado direita de uma produção.

A cada passo, deve-se manter controle sobre:

- Qual é o nó da árvore de derivação se está expandindo?
- Qual é a próxima *token* da sentença de entrada (***lookahead***)?
  - Isto ajuda a selecionar qual a regra que deve ser usada na substituição do não-terminal.

Parsing Recursivo Descendente

Objetivo

Determinar se uma sentença pode ser derivada a partir do símbolo da partida de uma gramática.

Abordagem

Substituir recursivamente um não-terminal pelo lado direita de uma produção.

A cada passo, deve-se manter controle sobre:

- Qual é o nó da árvore de derivação se está expandindo?
- Qual é a próxima *token* da sentença de entrada (***lookahead***)?
  - Isto ajuda a selecionar qual a regra que deve ser usada na substituição do não-terminal.

Parsing Recursivo Descendente

Objetivo

Determinar se uma sentença pode ser derivada a partir do símbolo da partida de uma gramática.

Abordagem

Substituir recursivamente um não-terminal pelo lado direita de uma produção.

A cada passo, deve-se manter controle sobre:

- Qual é o nó da árvore de derivação se está expandindo?
- Qual é a próxima *token* da sentença de entrada (***lookahead***)?
  - Isto ajuda a selecionar qual a regra que deve ser usada na substituição do não-terminal.

Parsing Recursivo Descendente

Abordagem (cont.)

A cada passo, existem 3 possibilidades:

1. Se estiver tentando casar um terminal:
   - Caso o ***lookahead*** seja o terminal desejado, então tem-se sucesso e a análise continua.
2. Se estiver tentando casar um não-terminal:
   - Seleciona qual regra aplicar baseada no ***lookahead***.
3. Caso contrário, o processo falha, gerando um erro de análise sintática.

Parsing Recursivo Descendente

Abordagem (cont.)

A cada passo, existem 3 possibilidades:

1. Se estiver tentando casar um terminal:
   - Caso o ***lookahead*** seja o terminal desejado, então tem-se sucesso e a análise continua.
2. Se estiver tentando casar um não-terminal:
   - Seleciona qual regra aplicar baseada no ***lookahead***.
3. Caso contrário, o processo falha, gerando um erro de análise sintática.

Parsing Recursivo Descendente

Abordagem (cont.)

A cada passo, existem 3 possibilidades:

1. Se estiver tentando casar um terminal:
   - Caso o ***lookahead*** seja o terminal desejado, então tem-se sucesso e a análise continua.
2. Se estiver tentando casar um não-terminal:
   - Seleciona qual regra aplicar baseada no ***lookahead***.
3. Caso contrário, o processo falha, gerando um erro de análise sintática.

Parsing Recursivo Descendente

Abordagem (cont.)

A cada passo, existem 3 possibilidades:

1. Se estiver tentando casar um terminal:
   - Caso o ***lookahead*** seja o terminal desejado, então tem-se sucesso e a análise continua.
2. Se estiver tentando casar um não-terminal:
   - Seleciona qual regra aplicar baseada no ***lookahead***.
3. Caso contrário, o processo falha, gerando um erro de análise sintática.

Parsing Recursivo Descendente

Abordagem (cont.)

A cada passo, existem 3 possibilidades:

1. Se estiver tentando casar um terminal:
   - Caso o ***lookahead*** seja o terminal desejado, então tem-se sucesso e a análise continua.
2. Se estiver tentando casar um não-terminal:
   - Seleciona qual regra aplicar baseada no ***lookahead***.
3. Caso contrário, o processo falha, gerando um erro de análise sintática.

Parsing Recursivo Descendente

Abordagem (cont.)

A cada passo, existem 3 possibilidades:

1. Se estiver tentando casar um terminal:
   - Caso o ***lookahead*** seja o terminal desejado, então tem-se sucesso e a análise continua.
2. Se estiver tentando casar um não-terminal:
   - Seleciona qual regra aplicar baseada no ***lookahead***.
3. Caso contrário, o processo falha, gerando um erro de análise sintática.

Backtracking

A forma mais simples (porém ineficiente) de se construir um *parser* descenden- te é utilizando retrocesso ("*backtracking*").

A ideia é tentar uma das regras de um não-terminal de cada vez até se tenha sucesso.

Caso seja impossível prosseguir devido a escolha de uma dada regra de um não-terminal (isto é, não se consiga produzir uma derivação que corresponda a sentença sob análise), deve-se selecionar outra regra para esse não-terminal e tentar novamente.

Backtracking

A forma mais simples (porém ineficiente) de se construir um *parser* descenden- te é utilizando retrocesso ("*backtracking*").

A ideia é tentar uma das regras de um não-terminal de cada vez até se tenha sucesso.

Caso seja impossível prosseguir devido a escolha de uma dada regra de um não-terminal (isto é, não se consiga produzir uma derivação que corresponda a sentença sob análise), deve-se selecionar outra regra para esse não-terminal e tentar novamente.

Backtracking

A forma mais simples (porém ineficiente) de se construir um *parser* descenden- te é utilizando retrocesso ("*backtracking*").

A ideia é tentar uma das regras de um não-terminal de cada vez até se tenha sucesso.

Caso seja impossível prosseguir devido a escolha de uma dada regra de um não-terminal (isto é, não se consiga produzir uma derivação que corresponda a sentença sob análise), deve-se selecionar outra regra para esse não-terminal e tentar novamente.

Backtracking

Considere GLC G: $S \rightarrow c\ A\ d$
$A \rightarrow a\ b \mid a$

Para a sentença cad, tem-se o seguinte processo de análise:

$$S \Rightarrow cAd \Rightarrow cabd$$

que não consegue produzir a sentença desejada.

Retrocedendo um passo e escolhendo outra regra para o não-terminal A, obtém-se sucesso:

$$S \Rightarrow cAd \Rightarrow cad$$

Backtracking

Considere GLC G: $S \rightarrow c\ A\ d$
$A \rightarrow a\ b \mid a$

Para a sentença *cad*, tem-se o seguinte processo de análise:

$$S \Rightarrow cAd \Rightarrow cabd$$

que não consegue produzir a sentença desejada.

Retrocedendo um passo e escolhendo outra regra para o não-terminal A, obtém-se sucesso:

$$S \Rightarrow cAd \Rightarrow cad$$

Backtracking

Considere GLC G: $S \rightarrow c\ A\ d$
$A \rightarrow a\ b \mid a$

Para a sentença *cad*, tem-se o seguinte processo de análise:

$$S \Rightarrow cAd \Rightarrow cabd$$

que não consegue produzir a sentença desejada.

Retrocedendo um passo e escolhendo outra regra para o não-terminal A, obtém-se sucesso:

$$S \Rightarrow cAd \Rightarrow cad$$

Backtracking

Considere GLC G: $S \rightarrow c\ A\ d$
$A \rightarrow a\ b \mid a$

Para a sentença *cad*, tem-se o seguinte processo de análise:

$$S \Rightarrow cAd \Rightarrow cabd$$

que não consegue produzir a sentença desejada.

Retrocedendo um passo e escolhendo outra regra para o não-terminal A, obtém-se sucesso:

$$S \Rightarrow cAd \Rightarrow cad$$

Backtracking

Considere GLC G: $S \rightarrow c\ A\ d$
$A \rightarrow a\ b \mid a$

Para a sentença *cad*, tem-se o seguinte processo de análise:

$$S \Rightarrow cAd \Rightarrow cabd$$

que não consegue produzir a sentença desejada.

Retrocedendo um passo e escolhendo outra regra para o não-terminal A, obtém-se sucesso:

$$S \Rightarrow cAd \Rightarrow cad$$

Backtracking

Considere GLC G: $S \rightarrow c\ A\ d$
$A \rightarrow a\ b \mid a$

Para a sentença *cad*, tem-se o seguinte processo de análise:

$$S \Rightarrow cAd \Rightarrow cabd$$

que não consegue produzir a sentença desejada.

Retrocedendo um passo e escolhendo outra regra para o não-terminal A, obtém-se sucesso:

$$S \Rightarrow cAd \Rightarrow cad$$

Lookahead

O problema básico da análise sintática é escolher qual produção utilizar em um dado momento da derivação.

Qualquer abordagem que utilize retrocesso geralmente resultará em algoritmos complexos e ineficientes se implementados em uma linguagem de programação convencional (procedural).

Para facilitar a implementação, pode-se introduzir a noção de "*lookahead*".

Lookahead representa uma tentativa de se analisar qual a possível produção deve ser aplicada, de forma a se escolher aquela que mais provavelmente irá resultar na derivação do(s) símbolo(s) corrente(s) na entrada.

Lookahead

O problema básico da análise sintática é escolher qual produção utilizar em um dado momento da derivação.

Qualquer abordagem que utilize retrocesso geralmente resultará em algoritmos complexos e ineficientes se implementados em uma linguagem de programação convencional (procedural).

Para facilitar a implementação, pode-se introduzir a noção de "*lookahead*".

Lookahead representa uma tentativa de se analisar qual a possível produção deve ser aplicada, de forma a se escolher aquela que mais provavelmente irá resultar na derivação do(s) símbolo(s) corrente(s) na entrada.

Lookahead

O problema básico da análise sintática é escolher qual produção utilizar em um dado momento da derivação.

Qualquer abordagem que utilize retrocesso geralmente resultará em algoritmos complexos e ineficientes se implementados em uma linguagem de programação convencional (procedural).

Para facilitar a implementação, pode-se introduzir a noção de "*lookahead*".

Lookahead representa uma tentativa de se analisar qual a possível produção deve ser aplicada, de forma a se escolher aquela que mais provavelmente irá resultar na derivação do(s) símbolo(s) corrente(s) na entrada.

Lookahead

O problema básico da análise sintática é escolher qual produção utilizar em um dado momento da derivação.

Qualquer abordagem que utilize retrocesso geralmente resultará em algoritmos complexos e ineficientes se implementados em uma linguagem de programação convencional (procedural).

Para facilitar a implementação, pode-se introduzir a noção de "*lookahead*".

Lookahead representa uma tentativa de se analisar qual a possível produção deve ser aplicada, de forma a se escolher aquela que mais provavelmente irá resultar na derivação do(s) símbolo(s) corrente(s) na entrada.

Lookahead

Suponha que se está no meio de um processo de derivação, em que:

- Forma sentencial atual: $\alpha X \beta$
- Sentença de entrada: $\alpha a \gamma$

de modo que:

- α é uma cadeia de terminais que corresponde à porção da entrada já verificada;
- X é o não-terminal mais à esquerda;
- β representa o restante da forma sentencial;
- a é o próximo símbolo da entrada e
- γ representa o restante da sentença de entrada.

Lookahead

Suponha que se está no meio de um processo de derivação, em que:

- Forma sentencial atual: $\alpha X \beta$
- Sentença de entrada: $\alpha a \gamma$

de modo que:

- α é uma cadeia de terminais que corresponde à porção da entrada já verificada;
- X é o não-terminal mais à esquerda;
- β representa o restante da forma sentencial;
- a é o próximo símbolo da entrada e
- γ representa o restante da sentença de entrada.

Lookahead

Suponha que se está no meio de um processo de derivação, em que:

- Forma sentencial atual: $\alpha X \beta$
- Sentença de entrada: $\alpha a \gamma$

de modo que:

- α é uma cadeia de terminais que corresponde a porção da entrada já verificada,
- X é o não-terminal mais à esquerda,
- β representa o restante da forma sentencial,
- a é o próximo símbolo da entrada e
- γ representa o restante da sentença de entrada.

Lookahead

Suponha que se está no meio de um processo de derivação, em que:

- Forma sentencial atual: $\alpha X \beta$
- Sentença de entrada: $\alpha a \gamma$

de modo que:

- α é uma cadeia de terminais que corresponde a porção da entrada já verificada,
- X é o não-terminal mais à esquerda,
- β representa o restante da forma sentencial,
- a é o próximo símbolo da entrada e
- γ representa o restante da sentença de entrada.

Lookahead

Suponha que se está no meio de um processo de derivação, em que:

- Forma sentencial atual: $\alpha X \beta$
- Sentença de entrada: $\alpha a \gamma$

de modo que:

- α é uma cadeia de terminais que corresponde a porção da entrada já verificada,
- X é o não-terminal mais à esquerda,
- β representa o restante da forma sentencial,
- a é o próximo símbolo da entrada e
- γ representa o restante da sentença de entrada.

Lookahead

Suponha que se está no meio de um processo de derivação, em que:

- Forma sentencial atual: $\alpha X \beta$
- Sentença de entrada: $\alpha a \gamma$

de modo que:

- α é uma cadeia de terminais que corresponde a porção da entrada já verificada,
- X é o não-terminal mais à esquerda,
- β representa o restante da forma sentencial,
- a é o próximo símbolo da entrada e
- γ representa o restante da sentença de entrada.

Lookahead

Suponha que se está no meio de um processo de derivação, em que:

- Forma sentencial atual: $\alpha X \beta$
- Sentença de entrada: $\alpha a \gamma$

de modo que:

- α é uma cadeia de terminais que corresponde a porção da entrada já verificada,
- X é o não-terminal mais à esquerda,
- β representa o restante da forma sentencial,
- a é o próximo símbolo da entrada e
- γ representa o restante da sentença de entrada.

Lookahead

Suponha que se está no meio de um processo de derivação, em que:

- Forma sentencial atual: $\alpha X \beta$
- Sentença de entrada: $\alpha a \gamma$

de modo que:

- α é uma cadeia de terminais que corresponde a porção da entrada já verificada,
- X é o não-terminal mais à esquerda,
- β representa o restante da forma sentencial,
- a é o próximo símbolo da entrada e
- γ representa o restante da sentença de entrada.

Lookahead

Suponha que se está no meio de um processo de derivação, em que:

- Forma sentencial atual: $\alpha X \beta$
- Sentença de entrada: $\alpha a \gamma$

de modo que:

- α é uma cadeia de terminais que corresponde a porção da entrada já verificada,
- X é o não-terminal mais à esquerda,
- β representa o restante da forma sentencial,
- a é o próximo símbolo da entrada e
- γ representa o restante da sentença de entrada.

Lookahead

Suponha que se está no meio de um processo de derivação, em que:

- Forma sentencial atual: $\alpha X \beta$
- Sentença de entrada: $\alpha a \gamma$

de modo que:

- α é uma cadeia de terminais que corresponde a porção da entrada já verificada,
- X é o não-terminal mais à esquerda,
- β representa o restante da forma sentencial,
- a é o próximo símbolo da entrada e
- γ representa o restante da sentença de entrada.

Deve-se determinar que regra será capaz de permitir que $X\beta$ derive $a\gamma$.

Lookahead

Para tanto, examina-se o lado direito das regras de X da seguinte forma:

1. Se o lado direito da regra iniciar com um terminal, ela deve ser da forma $X \rightarrow a\delta$ para poder ser útil
2. Se o lado direito da regra iniciar com um não-terminal, isto é, $X \rightarrow Y\delta$ então
3. Se for determinado que X pode produzir λ, então deve-se verificar quais símbolos podem ser derivados daqueles que sucedem X na forma sentencial (isto é, qual é o primeiro símbolo vindo após X)

Lookahead

Para tanto, examina-se o lado direito das regras de X da seguinte forma:

1. Se o lado direito da regra iniciar com um terminal, ela deve ser da forma $X \rightarrow a\delta$ para poder ser útil
2. Se o lado direito da regra iniciar com um não-terminal, isto é, $X \rightarrow Y\delta$ então
   - Deve-se examinar as regras de Y para ver se alguma delas pode derivar a
   - Caso Y possa produzir λ, deve-se também examinar δ, isto é, aplicar os passos 1 e 2 às regras de δ
3. Se for determinado que X pode produzir λ, então deve-se verificar quais símbolos podem ser derivados daqueles que seguem X na forma sentencial (isto é, qual é o primeiro símbolo derivado de β)

Lookahead

Para tanto, examina-se o lado direito das regras de X da seguinte forma:

1. Se o lado direito da regra iniciar com um terminal, ela deve ser da forma $X \rightarrow a\delta$ para poder ser útil
2. Se o lado direito da regra iniciar com um não-terminal, isto é, $X \rightarrow Y\delta$ então
   - Deve-se examinar as regras de Y para ver se alguma delas pode derivar a
   - Caso Y possa produzir λ, deve então examinar δ, isto é, aplicar os passos 1 e 2 as regras de δ
3. Se for determinado que X pode produzir λ, então deve-se verificar quais símbolos podem ser derivados daqueles que seguem X na forma sentencial (isto é, qual é o primeiro símbolo derivado de β)

Lookahead

Para tanto, examina-se o lado direito das regras de X da seguinte forma:

1. Se o lado direito da regra iniciar com um terminal, ela deve ser da forma $X \rightarrow a\delta$ para poder ser útil
2. Se o lado direito da regra iniciar com um não-terminal, isto é, $X \rightarrow Y\delta$ então
   - Deve-se examinar as regras de Y para ver se alguma delas pode derivar a
   - Caso Y possa produzir λ, deve então examinar δ, isto é, aplicar os passos 1 e 2 as regras de δ
3. Se for determinado que X pode produzir λ, então deve-se verificar quais símbolos podem ser derivados daqueles que seguem X na forma sentencial (isto é, qual é o primeiro símbolo derivado de β)

Lookahead

Para tanto, examina-se o lado direito das regras de X da seguinte forma:

1. Se o lado direito da regra iniciar com um terminal, ela deve ser da forma $X \rightarrow a\delta$ para poder ser útil
2. Se o lado direito da regra iniciar com um não-terminal, isto é, $X \rightarrow Y\delta$ então
   - Deve-se examinar as regras de Y para ver se alguma delas pode derivar a
   - Caso Y possa produzir λ, deve então examinar δ, isto é, aplicar os passos 1 e 2 as regras de δ
3. Se for determinado que X pode produzir λ, então deve-se verificar quais símbolos podem ser derivados daqueles que seguem X na forma sentencial (isto é, qual é o primeiro símbolo derivado de β)

Lookahead

Para tanto, examina-se o lado direito das regras de X da seguinte forma:

1. Se o lado direito da regra iniciar com um terminal, ela deve ser da forma $X \rightarrow a\delta$ para poder ser útil
2. Se o lado direito da regra iniciar com um não-terminal, isto é, $X \rightarrow Y\delta$ então
   - Deve-se examinar as regras de Y para ver se alguma delas pode derivar a
   - Caso Y possa produzir λ, deve então examinar δ, isto é, aplicar os passos 1 e 2 as regras de δ
3. Se for determinado que X pode produzir λ, então deve-se verificar quais símbolos podem ser derivados daqueles que seguem X na forma sentencial (isto é, qual é o primeiro símbolo derivado de β)

Backtracking × Predição

Backtracking

- Escolher alguma regra
- Se falhar, tentar uma regra diferente
- A análise falha se todas as escolhas falharem

Predição

- Analisar a gramática para determinar os conjuntos da função **FIRST**
- Usar ***lookahead*** para selecionar qual regra utilizar
- A análise falha se ***lookahead*** não corresponder a nenhum elemento de **FIRST**

OBS.: Se X for o não-terminal corrente (isto é, o mais à esquerda da forma sentencial corrente) e $\boldsymbol{a}$ for o próximo símbolo na entrada (representado pelo ***lookahead***), deve-se selecionar aquela regra de X cujo **FIRST** contenha $\boldsymbol{a}$.

Backtracking × Predição

Backtracking

- Escolher alguma regra
- Se falhar, tentar uma regra diferente
- A análise falha se todas as escolhas falharem

Predição

- Analisar a gramática para determinar os conjuntos da função **FIRST**
- Usar ***lookahead*** para selecionar qual regra utilizar
- A análise falha se ***lookahead*** não corresponder a nenhum elemento de **FIRST**

OBS.: Se X for o não-terminal corrente (isto é, o mais à esquerda da forma sentencial corrente) e ***a*** for o próximo símbolo na entrada (representado pelo ***lookahead***), deve-se selecionar aquela regra de X cujo **FIRST** contenha ***a***.

Backtracking × Predição

Backtracking

- Escolher alguma regra
- Se falhar, tentar uma regra diferente
- A análise falha se todas as escolhas falharem

Predição

- Analisar a gramática para determinar os conjuntos da função **FIRST**
- Usar ***lookahead*** para selecionar qual regra utilizar
- A análise falha se ***lookahead*** não corresponder a nenhum elemento de **FIRST**

OBS.: Se X for o não-terminal corrente (isto é, o mais à esquerda da forma sentencial corrente) e ***a*** for o próximo símbolo na entrada (representado pelo ***lookahead***), deve-se selecionar aquela regra de X cujo **FIRST** contenha ***a***.

Função FIRST

FIRST

Deve-se analisar o lado direito de todas as regras para se determinar todos os terminais que podem iniciar sentenças derivadas a partir dessas regras.

Define-se **FIRST** da seguinte forma:

- Para todo terminal a, $\textbf{FIRST}(a) = \{a\}$. Também vale para λ, $\textbf{FIRST}(\lambda) = \{\lambda\}$.
- Para todo não-terminal A com as seguintes regras $A \rightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$, $\textbf{FIRST}(A) = \textbf{FIRST}(\alpha_1) \cup \textbf{FIRST}(\alpha_2) \cup \ldots \textbf{FIRST}(\alpha_n)$
- Para todo lado direito α_i da forma $\beta_1\beta_2\ldots\beta_n$, tem-se:
  - $\textbf{FIRST}(\beta_1) - \{\lambda\}$ está em $\textbf{FIRST}(\beta_1\beta_2\ldots\beta_n)$
    Se β_1 pode derivar λ, então $\textbf{FIRST}(\beta_2) - \{\lambda\}$ também está em $\textbf{FIRST}(\beta_1\beta_2\ldots\beta_n)$
    Se β_1 e β_2 podem derivar λ, então $\textbf{FIRST}(\beta_3) - \{\lambda\}$ também está em $\textbf{FIRST}(\beta_1\beta_2\ldots\beta_n)$
    ...
    Se $\beta_1, \beta_2, \ldots, \beta_i$ podem derivar λ, então $\textbf{FIRST}(\beta_{i+1}) - \{\lambda\}$ também está em $\textbf{FIRST}(\beta_1\beta_2\ldots\beta_n)$
  - $\lambda \in \textbf{FIRST}(\beta_1\beta_2\ldots\beta_n)$ se e somente se $\lambda \in \textbf{FIRST}(\beta_i), \forall i = 1, \ldots, n$

Função FIRST

FIRST

Deve-se analisar o lado direito de todas as regras para se determinar todos os terminais que podem iniciar sentenças derivadas a partir dessas regras.
Define-se **FIRST** da seguinte forma:

- Para todo terminal a, $\textbf{FIRST}(a) = \{a\}$. Também vale para λ, $\textbf{FIRST}(\lambda) = \{\lambda\}$.
- Para todo não-terminal A com as seguintes regras $A \rightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$, $\textbf{FIRST}(A) = \textbf{FIRST}(\alpha_1) \cup \textbf{FIRST}(\alpha_2) \cup \ldots \textbf{FIRST}(\alpha_n)$
- Para todo lado direito α_i da forma $\beta_1\beta_2\ldots\beta_n$, tem-se:
  - $\textbf{FIRST}(\beta_1) - \{\lambda\}$ está em $\textbf{FIRST}(\beta_1\beta_2\ldots\beta_n)$
    Se β_1 pode derivar λ, então $\textbf{FIRST}(\beta_2) - \{\lambda\}$ também está em $\textbf{FIRST}(\beta_1\beta_2\ldots\beta_n)$
    Se β_1 e β_2 podem derivar λ, então $\textbf{FIRST}(\beta_3) - \{\lambda\}$ também está em $\textbf{FIRST}(\beta_1\beta_2\ldots\beta_n)$
    ...
    Se $\beta_1, \beta_2, \ldots, \beta_i$ podem derivar λ, então $\textbf{FIRST}(\beta_{i+1}) - \{\lambda\}$ também está em $\textbf{FIRST}(\beta_1\beta_2\ldots\beta_n)$
  - $\lambda \in \textbf{FIRST}(\beta_1\beta_2\ldots\beta_n)$ se e somente se $\lambda \in \textbf{FIRST}(\beta_i), \forall i = 1, \ldots, n$

Função FIRST

FIRST

Deve-se analisar o lado direito de todas as regras para se determinar todos os terminais que podem iniciar sentenças derivadas a partir dessas regras.
Define-se **FIRST** da seguinte forma:

- Para todo terminal a, **FIRST**$(a) = \{a\}$. Também vale para λ, **FIRST**$(\lambda) = \{\lambda\}$.
- Para todo não-terminal A com as seguintes regras $A \rightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$, **FIRST**$(A) =$ **FIRST**$(\alpha_1) \cup$ **FIRST**$(\alpha_2) \cup \ldots$ **FIRST**(α_n)
- Para todo lado direito α_i da forma $\beta_1\beta_2\ldots\beta_n$, tem-se:
  - **FIRST**$(\beta_1) - \{\lambda\}$ está em **FIRST**$(\beta_1\beta_2\ldots\beta_n)$
    Se β_1 pode derivar λ, então **FIRST**$(\beta_2) - \{\lambda\}$ também está em **FIRST**$(\beta_1\beta_2\ldots\beta_n)$
    Se β_1 e β_2 podem derivar λ, então **FIRST**$(\beta_3) - \{\lambda\}$ também está em **FIRST**$(\beta_1\beta_2\ldots\beta_n)$
    ...
    Se $\beta_1, \beta_2, \ldots, \beta_i$ podem derivar λ, então **FIRST**$(\beta_{i+1}) - \{\lambda\}$ também está em **FIRST**$(\beta_1\beta_2\ldots\beta_n)$
  - $\lambda \in$ **FIRST**$(\beta_1\beta_2\ldots\beta_n)$ se e somente se $\lambda \in$ **FIRST**$(\beta_i), \forall i = 1, \ldots, n$

Função FIRST

FIRST

Deve-se analisar o lado direito de todas as regras para se determinar todos os terminais que podem iniciar sentenças derivadas a partir dessas regras.
Define-se **FIRST** da seguinte forma:

- Para todo terminal a, **FIRST**$(a) = \{a\}$. Também vale para λ, **FIRST**$(\lambda) = \{\lambda\}$.
- Para todo não-terminal A com as seguintes regras $A \rightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$, **FIRST**$(A) =$ **FIRST**$(\alpha_1) \cup$ **FIRST**$(\alpha_2) \cup \ldots$ **FIRST**(α_n)
- Para todo lado direito α_i da forma $\beta_1\beta_2\ldots\beta_n$, tem-se:
  - **FIRST**$(\beta_1) - \{\lambda\}$ está em **FIRST**$(\beta_1\beta_2\ldots\beta_n)$
    Se β_1 pode derivar λ, então **FIRST**$(\beta_2) - \{\lambda\}$ também está em **FIRST**$(\beta_1\beta_2\ldots\beta_n)$
    Se β_1 e β_2 podem derivar λ, então **FIRST**$(\beta_3) - \{\lambda\}$ também está em **FIRST**$(\beta_1\beta_2\ldots\beta_n)$
    ...
    Se $\beta_1, \beta_2, \ldots, \beta_i$ podem derivar λ, então **FIRST**$(\beta_{i+1}) - \{\lambda\}$ também está em **FIRST**$(\beta_1\beta_2\ldots\beta_n)$
  - $\lambda \in$ **FIRST**$(\beta_1\beta_2\ldots\beta_n)$ se e somente se $\lambda \in$ **FIRST**$(\beta_i), \forall i = 1, \ldots, n$

Função FIRST

FIRST

Deve-se analisar o lado direito de todas as regras para se determinar todos os terminais que podem iniciar sentenças derivadas a partir dessas regras.
Define-se **FIRST** da seguinte forma:

- Para todo terminal a, **FIRST**$(a) = \{a\}$. Também vale para λ, **FIRST**$(\lambda) = \{\lambda\}$.
- Para todo não-terminal A com as seguintes regras $A \rightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$, **FIRST**$(A) =$ **FIRST**$(\alpha_1) \cup$ **FIRST**$(\alpha_2) \cup \ldots$ **FIRST**(α_n)
- Para todo lado direito α_i da forma $\beta_1\beta_2\ldots\beta_n$, tem-se:
  - **FIRST**$(\beta_1) - \{\lambda\}$ está em **FIRST**$(\beta_1\beta_2\ldots\beta_n)$
    Se β_1 pode derivar λ, então **FIRST**$(\beta_2) - \{\lambda\}$ também está em **FIRST**$(\beta_1\beta_2\ldots\beta_n)$
    Se β_1 e β_2 podem derivar λ, então **FIRST**$(\beta_3) - \{\lambda\}$ também está em **FIRST**$(\beta_1\beta_2\ldots\beta_n)$
    ...
    Se $\beta_1, \beta_2, \ldots, \beta_i$ podem derivar λ, então **FIRST**$(\beta_{i+1}) - \{\lambda\}$ também está em **FIRST**$(\beta_1\beta_2\ldots\beta_n)$
  - $\lambda \in$ **FIRST**$(\beta_1\beta_2\ldots\beta_n)$ se e somente se $\lambda \in$ **FIRST**$(\beta_i), \forall i = 1, \ldots, n$

Função FIRST

FIRST

Deve-se analisar o lado direito de todas as regras para se determinar todos os terminais que podem iniciar sentenças derivadas a partir dessas regras.
Define-se **FIRST** da seguinte forma:

- Para todo terminal a, **FIRST**$(a) = \{a\}$. Também vale para λ, **FIRST**$(\lambda) = \{\lambda\}$.
- Para todo não-terminal A com as seguintes regras $A \rightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$, **FIRST**$(A) =$ **FIRST**$(\alpha_1) \cup$ **FIRST**$(\alpha_2) \cup \ldots$ **FIRST**(α_n)
- Para todo lado direito α_i da forma $\beta_1\beta_2 \ldots \beta_n$, tem-se:
  - **FIRST**$(\beta_1) - \{\lambda\}$ está em **FIRST**$(\beta_1\beta_2 \ldots \beta_n)$
    Se β_1 pode derivar λ, então **FIRST**$(\beta_2) - \{\lambda\}$ também está em **FIRST**$(\beta_1\beta_2 \ldots \beta_n)$
    Se β_1 e β_2 podem derivar λ, então **FIRST**$(\beta_3) - \{\lambda\}$ também está em **FIRST**$(\beta_1\beta_2 \ldots \beta_n)$
    ...
    Se $\beta_1, \beta_2, \ldots, \beta_i$ podem derivar λ, então **FIRST**$(\beta_{i+1}) - \{\lambda\}$ também está em **FIRST**$(\beta_1\beta_2 \ldots \beta_n)$
  - $\lambda \in$ **FIRST**$(\beta_1\beta_2 \ldots \beta_n)$ se e somente se $\lambda \in$ **FIRST**$(\beta_i), \forall i = 1, \ldots, n$

Função FIRST

FIRST

Deve-se analisar o lado direito de todas as regras para se determinar todos os terminais que podem iniciar sentenças derivadas a partir dessas regras.
Define-se **FIRST** da seguinte forma:

- Para todo terminal a, **FIRST**$(a) = \{a\}$. Também vale para λ, **FIRST**$(\lambda) = \{\lambda\}$.
- Para todo não-terminal A com as seguintes regras $A \rightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$, **FIRST**$(A) =$ **FIRST**$(\alpha_1) \cup$ **FIRST**$(\alpha_2) \cup \ldots$ **FIRST**(α_n)
- Para todo lado direito α_i da forma $\beta_1\beta_2 \ldots \beta_n$, tem-se:
  - **FIRST**$(\beta_1) - \{\lambda\}$ está em **FIRST**$(\beta_1\beta_2 \ldots \beta_n)$
    Se β_1 pode derivar λ, então **FIRST**$(\beta_2) - \{\lambda\}$ também está em **FIRST**$(\beta_1\beta_2 \ldots \beta_n)$
    Se β_1 e β_2 podem derivar λ, então **FIRST**$(\beta_3) - \{\lambda\}$ também está em **FIRST**$(\beta_1\beta_2 \ldots \beta_n)$
    ...
    Se $\beta_1, \beta_2, \ldots, \beta_i$ podem derivar λ, então **FIRST**$(\beta_{i+1}) - \{\lambda\}$ também está em **FIRST**$(\beta_1\beta_2 \ldots \beta_n)$
  - $\lambda \in$ **FIRST**$(\beta_1\beta_2 \ldots \beta_n)$ se e somente se $\lambda \in$ **FIRST**$(\beta_i), \forall i = 1, \ldots, n$

Função FIRST

FIRST

Deve-se analisar o lado direito de todas as regras para se determinar todos os terminais que podem iniciar sentenças derivadas a partir dessas regras.
Define-se **FIRST** da seguinte forma:

- Para todo terminal a, **FIRST**$(a) = \{a\}$. Também vale para λ, **FIRST**$(\lambda) = \{\lambda\}$.
- Para todo não-terminal A com as seguintes regras $A \rightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$, **FIRST**$(A) =$ **FIRST**$(\alpha_1) \cup$ **FIRST**$(\alpha_2) \cup \ldots$ **FIRST**(α_n)
- Para todo lado direito α_i da forma $\beta_1\beta_2 \ldots \beta_n$, tem-se:
  - **FIRST**$(\beta_1) - \{\lambda\}$ está em **FIRST**$(\beta_1\beta_2 \ldots \beta_n)$
    Se β_1 pode derivar λ, então **FIRST**$(\beta_2) - \{\lambda\}$ também está em **FIRST**$(\beta_1\beta_2 \ldots \beta_n)$
    Se β_1 e β_2 podem derivar λ, então **FIRST**$(\beta_3) - \{\lambda\}$ também está em **FIRST**$(\beta_1\beta_2 \ldots \beta_n)$
    ...
    Se $\beta_1, \beta_2, \ldots, \beta_i$ podem derivar λ, então **FIRST**$(\beta_{i+1}) - \{\lambda\}$ também está em **FIRST**$(\beta_1\beta_2 \ldots \beta_n)$
  - $\lambda \in$ **FIRST**$(\beta_1\beta_2 \ldots \beta_n)$ se e somente se $\lambda \in$ **FIRST**$(\beta_i), \forall i = 1, \ldots, n$

Função FIRST

FIRST

Deve-se analisar o lado direito de todas as regras para se determinar todos os terminais que podem iniciar sentenças derivadas a partir dessas regras.
Define-se **FIRST** da seguinte forma:

- Para todo terminal a, **FIRST**$(a) = \{a\}$. Também vale para λ, **FIRST**$(\lambda) = \{\lambda\}$.
- Para todo não-terminal A com as seguintes regras $A \rightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$, **FIRST**$(A) =$ **FIRST**$(\alpha_1) \cup$ **FIRST**$(\alpha_2) \cup \ldots$ **FIRST**(α_n)
- Para todo lado direito α_i da forma $\beta_1\beta_2\ldots\beta_n$, tem-se:
  - **FIRST**$(\beta_1) - \{\lambda\}$ está em **FIRST**$(\beta_1\beta_2\ldots\beta_n)$
    Se β_1 pode derivar λ, então **FIRST**$(\beta_2) - \{\lambda\}$ também está em **FIRST**$(\beta_1\beta_2\ldots\beta_n)$
    Se β_1 e β_2 podem derivar λ, então **FIRST**$(\beta_3) - \{\lambda\}$ também está em **FIRST**$(\beta_1\beta_2\ldots\beta_n)$
    ...
    Se $\beta_1, \beta_2, \ldots, \beta_i$ podem derivar λ, então **FIRST**$(\beta_{i+1}) - \{\lambda\}$ também está em **FIRST**$(\beta_1\beta_2\ldots\beta_n)$
  - $\lambda \in$ **FIRST**$(\beta_1\beta_2\ldots\beta_n)$ se e somente se $\lambda \in$ **FIRST**$(\beta_i), \forall i = 1, \ldots, n$

Função FIRST

FIRST

Deve-se analisar o lado direito de todas as regras para se determinar todos os terminais que podem iniciar sentenças derivadas a partir dessas regras.
Define-se **FIRST** da seguinte forma:

- Para todo terminal a, **FIRST**$(a) = \{a\}$. Também vale para λ, **FIRST**$(\lambda) = \{\lambda\}$.
- Para todo não-terminal A com as seguintes regras $A \rightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$, **FIRST**$(A) =$ **FIRST**$(\alpha_1) \cup$ **FIRST**$(\alpha_2) \cup \ldots$ **FIRST**(α_n)
- Para todo lado direito α_i da forma $\beta_1\beta_2\ldots\beta_n$, tem-se:
  - **FIRST**$(\beta_1) - \{\lambda\}$ está em **FIRST**$(\beta_1\beta_2\ldots\beta_n)$
    Se β_1 pode derivar λ, então **FIRST**$(\beta_2) - \{\lambda\}$ também está em **FIRST**$(\beta_1\beta_2\ldots\beta_n)$
    Se β_1 e β_2 podem derivar λ, então **FIRST**$(\beta_3) - \{\lambda\}$ também está em **FIRST**$(\beta_1\beta_2\ldots\beta_n)$
    ...
    Se $\beta_1, \beta_2, \ldots, \beta_i$ podem derivar λ, então **FIRST**$(\beta_{i+1}) - \{\lambda\}$ também está em **FIRST**$(\beta_1\beta_2\ldots\beta_n)$
  - $\lambda \in$ **FIRST**$(\beta_1\beta_2\ldots\beta_n)$ se e somente se $\lambda \in$ **FIRST**$(\beta_i), \forall i = 1, \ldots, n$

Função FIRST

FIRST – Exemplo 1

Considere GLC G: $S \rightarrow xyz \mid abc$

$\mathbf{FIRST}(x) = \{x\}, \mathbf{FIRST}(y) = \{y\}, \mathbf{FIRST}(z) = \{z\}$
$\mathbf{FIRST}(a) = \{a\}, \mathbf{FIRST}(b) = \{b\}, \mathbf{FIRST}(c) = \{c\}$

$\mathbf{FIRST}(xyz) = \mathbf{FIRST}(x) = \{x\}$, $\mathbf{FIRST}(abc) = \mathbf{FIRST}(a) = \{a\}$

$\mathbf{FIRST}(S) = \mathbf{FIRST}(xyz) \cup \mathbf{FIRST}(abc) = \{x, a\}$

Função FIRST

FIRST – Exemplo 1

Considere GLC G: $S \rightarrow xyz \mid abc$

$\textbf{FIRST}(x) = \{x\}, \textbf{FIRST}(y) = \{y\}, \textbf{FIRST}(z) = \{z\}$
$\textbf{FIRST}(a) = \{a\}, \textbf{FIRST}(b) = \{b\}, \textbf{FIRST}(c) = \{c\}$

$\textbf{FIRST}(xyz) = \textbf{FIRST}(x) = \{x\}$, $\textbf{FIRST}(abc) = \textbf{FIRST}(a) = \{a\}$

$\textbf{FIRST}(S) = \textbf{FIRST}(xyz) \cup \textbf{FIRST}(abc) = \{x, a\}$

Função FIRST

FIRST – Exemplo 1

Considere GLC G: $S \rightarrow xyz \mid abc$

$\mathbf{FIRST}(x) = \{x\}, \mathbf{FIRST}(y) = \{y\}, \mathbf{FIRST}(z) = \{z\}$
$\mathbf{FIRST}(a) = \{a\}, \mathbf{FIRST}(b) = \{b\}, \mathbf{FIRST}(c) = \{c\}$

$\mathbf{FIRST}(xyz) = \mathbf{FIRST}(x) = \{x\}$, $\mathbf{FIRST}(abc) = \mathbf{FIRST}(a) = \{a\}$

$\mathbf{FIRST}(S) = \mathbf{FIRST}(xyz) \cup \mathbf{FIRST}(abc) = \{x, a\}$

Função FIRST

FIRST – Exemplo 1

Considere GLC G: $S \rightarrow xyz \mid abc$

$\mathbf{FIRST}(x) = \{x\}, \mathbf{FIRST}(y) = \{y\}, \mathbf{FIRST}(z) = \{z\}$
$\mathbf{FIRST}(a) = \{a\}, \mathbf{FIRST}(b) = \{b\}, \mathbf{FIRST}(c) = \{c\}$

$\mathbf{FIRST}(xyz) = \mathbf{FIRST}(x) = \{x\}$, $\mathbf{FIRST}(abc) = \mathbf{FIRST}(a) = \{a\}$

$\mathbf{FIRST}(S) = \mathbf{FIRST}(xyz) \cup \mathbf{FIRST}(abc) = \{x, a\}$

Função FIRST

FIRST – Exemplo 1

Considere GLC G: $S \rightarrow xyz \mid abc$

$\mathbf{FIRST}(x) = \{x\}, \mathbf{FIRST}(y) = \{y\}, \mathbf{FIRST}(z) = \{z\}$
$\mathbf{FIRST}(a) = \{a\}, \mathbf{FIRST}(b) = \{b\}, \mathbf{FIRST}(c) = \{c\}$

$\mathbf{FIRST}(xyz) = \mathbf{FIRST}(x) = \{x\}$, $\mathbf{FIRST}(abc) = \mathbf{FIRST}(a) = \{a\}$

$\mathbf{FIRST}(S) = \mathbf{FIRST}(xyz) \cup \mathbf{FIRST}(abc) = \{x, a\}$

Função FIRST

FIRST – Exemplo 2

Considere GLC G: $S \to A \mid B$
$A \to x \mid y$
$B \to z$

$\mathbf{FIRST}(x) = \{x\}, \mathbf{FIRST}(y) = \{y\}, \mathbf{FIRST}(z) = \{z\}$

$\mathbf{FIRST}(A) = \mathbf{FIRST}(x) \cup \mathbf{FIRST}(y) = \{x, y\}$

$\mathbf{FIRST}(B) = \mathbf{FIRST}(z) = \{z\}$

$\mathbf{FIRST}(S) = \mathbf{FIRST}(A) \cup \mathbf{FIRST}(B) = \{x, y, z\}$

Função FIRST

FIRST – Exemplo 2

Considere GLC G: $S \to A \mid B$
$A \to x \mid y$
$B \to z$

$\mathbf{FIRST}(x) = \{x\}, \mathbf{FIRST}(y) = \{y\}, \mathbf{FIRST}(z) = \{z\}$

$\mathbf{FIRST}(A) = \mathbf{FIRST}(x) \cup \mathbf{FIRST}(y) = \{x, y\}$

$\mathbf{FIRST}(B) = \mathbf{FIRST}(z) = \{z\}$

$\mathbf{FIRST}(S) = \mathbf{FIRST}(A) \cup \mathbf{FIRST}(B) = \{x, y, z\}$

Função FIRST

FIRST – Exemplo 2

Considere GLC G: $S \to A \mid B$
$A \to x \mid y$
$B \to z$

$\mathbf{FIRST}(x) = \{x\}, \mathbf{FIRST}(y) = \{y\}, \mathbf{FIRST}(z) = \{z\}$

$\mathbf{FIRST}(A) = \mathbf{FIRST}(x) \cup \mathbf{FIRST}(y) = \{x, y\}$

$\mathbf{FIRST}(B) = \mathbf{FIRST}(z) = \{z\}$

$\mathbf{FIRST}(S) = \mathbf{FIRST}(A) \cup \mathbf{FIRST}(B) = \{x, y, z\}$

Função FIRST

FIRST – Exemplo 2

Considere GLC G: $S \to A \mid B$
$A \to x \mid y$
$B \to z$

$\mathbf{FIRST}(x) = \{x\}, \mathbf{FIRST}(y) = \{y\}, \mathbf{FIRST}(z) = \{z\}$

$\mathbf{FIRST}(A) = \mathbf{FIRST}(x) \cup \mathbf{FIRST}(y) = \{x, y\}$

$\mathbf{FIRST}(B) = \mathbf{FIRST}(z) = \{z\}$

$\mathbf{FIRST}(S) = \mathbf{FIRST}(A) \cup \mathbf{FIRST}(B) = \{x, y, z\}$

Função FIRST

FIRST – Exemplo 2

Considere GLC G: $S \to A \mid B$
$A \to x \mid y$
$B \to z$

$\mathbf{FIRST}(x) = \{x\}, \mathbf{FIRST}(y) = \{y\}, \mathbf{FIRST}(z) = \{z\}$

$\mathbf{FIRST}(A) = \mathbf{FIRST}(x) \cup \mathbf{FIRST}(y) = \{x, y\}$

$\mathbf{FIRST}(B) = \mathbf{FIRST}(z) = \{z\}$

$\mathbf{FIRST}(S) = \mathbf{FIRST}(A) \cup \mathbf{FIRST}(B) = \{x, y, z\}$

Função FIRST

FIRST – Exemplo 3

Considere GLC G: $E \rightarrow \mathbf{id} = \mathbf{n} \mid \{\ L\ \}$
$L \rightarrow\ E\ ;\ L \mid \lambda$

$\mathbf{FIRST}(\mathbf{id}) = \{\mathbf{id}\}$, $\mathbf{FIRST}('=') = \{'='\}$, $\mathbf{FIRST}(\mathbf{n}) = \{\mathbf{n}\}$,
$\mathbf{FIRST}('\{') = \{'\{'\}$, $\mathbf{FIRST}('\}') = \{'\}'\}$, $\mathbf{FIRST}(';') = \{';'\}$

$\mathbf{FIRST}(E) = \mathbf{FIRST}(\mathbf{id} = \mathbf{n}) \cup \mathbf{FIRST}(\{\ L\ \}) = \{\mathbf{id},\ '\{'\}$

$\mathbf{FIRST}(L) = \mathbf{FIRST}(E\ ;\ L) \cup \mathbf{FIRST}(\lambda) = \{\mathbf{id},\ '\{', \lambda\}$

Função FIRST

FIRST – Exemplo 3

Considere GLC G: $E \rightarrow \mathbf{id} = \mathbf{n} \mid \{\ L\ \}$
$L \rightarrow\ E\ ;\ L \mid \lambda$

$\mathbf{FIRST}(\mathbf{id}) = \{\mathbf{id}\}$, $\mathbf{FIRST}('=') = \{'='\}$, $\mathbf{FIRST}(\mathbf{n}) = \{\mathbf{n}\}$,
$\mathbf{FIRST}('\{') = \{'\{'\}$, $\mathbf{FIRST}('\}') = \{'\}'\}$, $\mathbf{FIRST}(';') = \{';'\}$

$\mathbf{FIRST}(E) = \mathbf{FIRST}(\mathbf{id} = \mathbf{n}) \cup \mathbf{FIRST}(\{\ L\ \}) = \{\mathbf{id}, '\{'\}$

$\mathbf{FIRST}(L) = \mathbf{FIRST}(E\ ;\ L) \cup \mathbf{FIRST}(\lambda) = \{\mathbf{id}, '\{', \lambda\}$

Função FIRST

FIRST – Exemplo 3

Considere GLC G: $E \rightarrow \mathbf{id} = \mathbf{n} \mid \{\ L\ \}$
$L \rightarrow\ E\ ;\ L \mid \lambda$

$\mathbf{FIRST}(\mathbf{id}) = \{\mathbf{id}\}$, $\mathbf{FIRST}('=') = \{'='\}$, $\mathbf{FIRST}(\mathbf{n}) = \{\mathbf{n}\}$,
$\mathbf{FIRST}('\{') = \{'\{'\}$, $\mathbf{FIRST}('\}') = \{'\}'\}$, $\mathbf{FIRST}(';') = \{';'\}$

$\mathbf{FIRST}(E) = \mathbf{FIRST}(\mathbf{id} = \mathbf{n}) \cup \mathbf{FIRST}(\{\ L\ \}) = \{\mathbf{id},\ '\{'\}$

$\mathbf{FIRST}(L) = \mathbf{FIRST}(E\ ;\ L) \cup \mathbf{FIRST}(\lambda) = \{\mathbf{id},\ '\{',\ \lambda\}$

Função FIRST

FIRST – Exemplo 3

Considere GLC G: $E \rightarrow \mathbf{id} = \mathbf{n} \mid \{\ L\ \}$
$L \rightarrow\ E\ ;\ L \mid \lambda$

$\mathbf{FIRST}(\mathbf{id}) = \{\mathbf{id}\}$, $\mathbf{FIRST}('=') = \{'='\}$, $\mathbf{FIRST}(\mathbf{n}) = \{\mathbf{n}\}$,
$\mathbf{FIRST}('\{') = \{'\{'\}$, $\mathbf{FIRST}('\}') = \{'\}'\}$, $\mathbf{FIRST}(';') = \{';'\}$

$\mathbf{FIRST}(E) = \mathbf{FIRST}(\mathbf{id} = \mathbf{n}) \cup \mathbf{FIRST}(\{\ L\ \}) = \{\mathbf{id},\ '\{'\}$

$\mathbf{FIRST}(L) = \mathbf{FIRST}(E\ ;\ L) \cup \mathbf{FIRST}(\lambda) = \{\mathbf{id},\ '\{', \lambda\}$

Função FIRST

FIRST – Exemplo 4

Considere GLC G: $E \rightarrow \mathbf{id} = \mathbf{n} \mid \{\ L\ \} \mid \lambda$
$L \rightarrow E\ ;\ L$

$\mathbf{FIRST}(\mathbf{id}) = \{\mathbf{id}\}$, $\mathbf{FIRST}('=') = \{'='\}$, $\mathbf{FIRST}(\mathbf{n}) = \{\mathbf{n}\}$,
$\mathbf{FIRST}('\{') = \{'\{'\}$, $\mathbf{FIRST}('\}') = \{'\}'\}$, $\mathbf{FIRST}(';') = \{';'\}$

$\mathbf{FIRST}(E) = \mathbf{FIRST}(\mathbf{id} = \mathbf{n}) \cup \mathbf{FIRST}(\{\ L\ \}) \cup \mathbf{FIRST}(\lambda) = \{\mathbf{id},\ '\{',\ \boldsymbol{\lambda}\}$

$\mathbf{FIRST}(L) = \mathbf{FIRST}(E\ ;\ L) = \{\mathbf{id},\ '\{',\ ';'\}$

Função FIRST

FIRST – Exemplo 4

Considere GLC G: $E \rightarrow \mathbf{id} = \mathbf{n} \mid \{\ L\ \} \mid \lambda$
$L \rightarrow E\ ;\ L$

$\mathbf{FIRST}(\mathbf{id}) = \{\mathbf{id}\}$, $\mathbf{FIRST}('=') = \{'='\}$, $\mathbf{FIRST}(\mathbf{n}) = \{\mathbf{n}\}$,
$\mathbf{FIRST}('\{') = \{'\{'\}$, $\mathbf{FIRST}('\}') = \{'\}'\}$, $\mathbf{FIRST}(';') = \{';'\}$

$\mathbf{FIRST}(E) = \mathbf{FIRST}(\mathbf{id} = \mathbf{n}) \cup \mathbf{FIRST}(\{\ L\ \}) \cup \mathbf{FIRST}(\lambda) = \{\mathbf{id},\ '\{',\ \boldsymbol{\lambda}\}$

$\mathbf{FIRST}(L) = \mathbf{FIRST}(E\ ;\ L) = \{\mathbf{id},\ '\{',\ ';'\}$

Função FIRST

FIRST – Exemplo 4

Considere GLC G: $E \rightarrow \mathbf{id} = \mathbf{n} \mid \{\ L\ \} \mid \lambda$
$L \rightarrow\ E\ ;\ L$

$\mathbf{FIRST}(\mathbf{id}) = \{\mathbf{id}\}$, $\mathbf{FIRST}('=') = \{'='\}$, $\mathbf{FIRST}(\mathbf{n}) = \{\mathbf{n}\}$,
$\mathbf{FIRST}('\{') = \{'\{'\}$, $\mathbf{FIRST}('\}') = \{'\}'\}$, $\mathbf{FIRST}(';') = \{';'\}$

$\mathbf{FIRST}(E) = \mathbf{FIRST}(\mathbf{id} = \mathbf{n}) \cup \mathbf{FIRST}(\{\ L\ \}) \cup \mathbf{FIRST}(\lambda) = \{\mathbf{id},\ '\{',\ \boldsymbol{\lambda}\}$

$\mathbf{FIRST}(L) = \mathbf{FIRST}(E\ ;\ L) = \{\mathbf{id},\ '\{',\ ';'\}$

Função FIRST

FIRST – Exemplo 4

Considere GLC G: $E \rightarrow \mathbf{id} = \mathbf{n} \mid \{\ L\ \} \mid \lambda$
$L \rightarrow E\ ;\ L$

$\mathbf{FIRST}(\mathbf{id}) = \{\mathbf{id}\}$, $\mathbf{FIRST}('=') = \{'='\}$, $\mathbf{FIRST}(\mathbf{n}) = \{\mathbf{n}\}$,
$\mathbf{FIRST}('\{') = \{'\{'\}$, $\mathbf{FIRST}('\}') = \{'\}'\}$, $\mathbf{FIRST}(';') = \{';'\}$

$\mathbf{FIRST}(E) = \mathbf{FIRST}(\mathbf{id} = \mathbf{n}) \cup \mathbf{FIRST}(\{\ L\ \}) \cup \mathbf{FIRST}(\lambda) = \{\mathbf{id},\ '\{',\ \boldsymbol{\lambda}\}$

$\mathbf{FIRST}(L) = \mathbf{FIRST}(E\ ;\ L) = \{\mathbf{id},\ '\{',\ \boldsymbol{';'}\}$

Parsing Preditivo Recursivo

Implementação de Parser Recursivo Descendente

- Para cada terminal a, cria-se uma função **match**(a)
  - Se o ***lookahead*** for a, a função consome o ***lookahead***, avança o ***lookahead*** para a próxima *token* e retorna.
  - Caso contrário, a função falha com um erro de análise.
- Para cada não-terminal X, cria-se uma função **parse_X**
  - Essa função é chamada quando se deseja analisar parte da entrada que corresponda (ou possa corresponder) a uma sentença derivada a partir de X.
  - A função **parse_S** criada para o símbolo de partida S é responsável por iniciar o processo de análise.

Parsing Preditivo Recursivo

Implementação de Parser Recursivo Descendente

- Para cada terminal a, cria-se uma função **match**(a)
  - Se o ***lookahead*** for a, a função consome o ***lookahead***, avança o ***lookahead*** para a próxima *token* e retorna.
  - Caso contrário, a função falha com um erro de análise.
- Para cada não-terminal X, cria-se uma função **parse_X**
  - Essa função é chamada quando se deseja analisar parte da entrada que corresponda (ou possa corresponder) a uma sentença derivada a partir de X.
  - A função **parse_S** criada para o símbolo de partida S é responsável por iniciar o processo de análise.

Parsing Preditivo Recursivo

Implementação de Parser Recursivo Descendente

- Para cada terminal a, cria-se uma função **match**(a)
  - Se o ***lookahead*** for a, a função consome o ***lookahead***, avança o ***lookahead*** para a próxima *token* e retorna.
  - Caso contrário, a função falha com um erro de análise.
- Para cada não-terminal X, cria-se uma função **parse_X**
  - Essa função é chamada quando se deseja analisar parte da entrada que corresponda (ou possa corresponder) a uma sentença derivada a partir de X.
  - A função **parse_S** criada para o símbolo de partida S é responsável por iniciar o processo de análise.

Parsing Preditivo Recursivo

Implementação de Parser Recursivo Descendente

- Para cada terminal a, cria-se uma função **match**(a)
  - Se o ***lookahead*** for a, a função consome o ***lookahead***, avança o ***lookahead*** para a próxima *token* e retorna.
  - Caso contrário, a função falha com um erro de análise.
- Para cada não-terminal X, cria-se uma função **parse_X**
  - Essa função é chamada quando se deseja analisar parte da entrada que corresponda (ou possa corresponder) a uma sentença derivada a partir de X.
  - A função **parse_S** criada para o símbolo de partida S é responsável por iniciar o processo de análise.

Parsing Preditivo Recursivo

Implementação de Parser Recursivo Descendente

- Para cada terminal a, cria-se uma função **match**(a)
  - Se o ***lookahead*** for a, a função consome o ***lookahead***, avança o ***lookahead*** para a próxima *token* e retorna.
  - Caso contrário, a função falha com um erro de análise.
- Para cada não-terminal X, cria-se uma função **parse_X**
  - Essa função é chamada quando se deseja analisar parte da entrada que corresponda (ou possa corresponder) a uma sentença derivada a partir de X.
  - A função **parse_S** criada para o símbolo de partida S é responsável por iniciar o processo de análise.

Parsing Preditivo Recursivo

Implementação de Parser Recursivo Descendente

- Para cada terminal a, cria-se uma função **match**(a)
  - Se o ***lookahead*** for a, a função consome o ***lookahead***, avança o ***lookahead*** para a próxima *token* e retorna.
  - Caso contrário, a função falha com um erro de análise.
- Para cada não-terminal X, cria-se uma função **parse_X**
  - Essa função é chamada quando se deseja analisar parte da entrada que corresponda (ou possa corresponder) a uma sentença derivada a partir de X.
  - A função **parse_S** criada para o símbolo de partida S é responsável por iniciar o processo de análise.

Parsing Preditivo Recursivo

Implementação de Parser Recursivo Descendente

- Para cada terminal a, cria-se uma função **match**(a)
  - Se o ***lookahead*** for a, a função consome o ***lookahead***, avança o ***lookahead*** para a próxima *token* e retorna.
  - Caso contrário, a função falha com um erro de análise.
- Para cada não-terminal X, cria-se uma função **parse_X**
  - Essa função é chamada quando se deseja analisar parte da entrada que corresponda (ou possa corresponder) a uma sentença derivada a partir de X.
  - A função **parse_S** criada para o símbolo de partida S é responsável por iniciar o processo de análise.

Parsing Preditivo Recursivo

Implementação de Parser Recursivo Descendente (cont.)

- A função **parse_X** faz o seguinte:
  - Considere que existem as regras $X \to \beta_1 \mid \ldots \mid \beta_k$ para o não-terminal X.
  - Selecionar a regra $X \to \beta_i$ tal que o ***lookahead*** $\in$ **FIRST**(β_i).
    - Deve-se garantir que **FIRST**$(\beta_i) \cap$ **FIRST**$(\beta_j) = \emptyset$, $\forall i \neq j$.
    - Caso não exista tal regra, mas exista a regra $X \to \lambda$, basta retornar.
    - Caso contrário, a função falha com um erro de análise.
  - Suponha que $\beta_i = \alpha_1 \ldots \alpha_n$. Chamar **parse_**$\alpha_1()$; $\ldots$ **parse_**$\alpha_n()$; e retornar. Caso α_j seja um terminal usar **match**(α_j).

Parsing Preditivo Recursivo

Implementação de Parser Recursivo Descendente (cont.)

- A função **parse_X** faz o seguinte:
  - Considere que existem as regras $X \to \beta_1 \mid \ldots \mid \beta_k$ para o não-terminal X.
  - Selecionar a regra $X \to \beta_i$ tal que o ***lookahead*** $\in$ **FIRST**(β_i).
    - Deve-se garantir que **FIRST**$(\beta_i) \cap$ **FIRST**$(\beta_j) = \emptyset$, $\forall i \neq j$.
    - Caso não exista tal regra, mas exista a regra $X \to \lambda$, basta retornar.
    - Caso contrário, a função falha com um erro de análise.
  - Suponha que $\beta_i = \alpha_1 \ldots \alpha_n$. Chamar **parse**_$\alpha_1()$; ... **parse**_$\alpha_n()$; e retornar. Caso α_j seja um terminal usar **match**(α_j).

Parsing Preditivo Recursivo

Implementação de Parser Recursivo Descendente (cont.)

- A função **parse_X** faz o seguinte:
  - Considere que existem as regras $X \to \beta_1 \mid \ldots \mid \beta_k$ para o não-terminal X.
  - Selecionar a regra $X \to \beta_i$ tal que o ***lookahead*** $\in$ **FIRST**(β_i).
    - Deve-se garantir que **FIRST**$(\beta_i) \cap$ **FIRST**$(\beta_j) = \emptyset$, $\forall i \neq j$.
    - Caso não exista tal regra, mas exista a regra $X \to \lambda$, basta retornar.
    - Caso contrário, a função falha com um erro de análise.
  - Suponha que $\beta_i = \alpha_1 \ldots \alpha_n$. Chamar **parse**_$\alpha_1()$; ... **parse**_$\alpha_n()$; e retornar. Caso α_j seja um terminal usar **match**(α_j).

Parsing Preditivo Recursivo

Implementação de Parser Recursivo Descendente (cont.)

- A função **parse_X** faz o seguinte:
  - Considere que existem as regras $X \to \beta_1 \mid \ldots \mid \beta_k$ para o não-terminal X.
  - Selecionar a regra $X \to \beta_i$ tal que o ***lookahead*** $\in$ **FIRST**(β_i).
    - Deve-se garantir que **FIRST**$(\beta_i) \cap$ **FIRST**$(\beta_j) = \emptyset$, $\forall i \neq j$.
    - Caso não exista tal regra, mas exista a regra $X \to \lambda$, basta retornar.
    - Caso contrário, a função falha com um erro de análise.
  - Suponha que $\beta_i = \alpha_1 \ldots \alpha_n$. Chamar **parse**_$\alpha_1()$; ... **parse**_$\alpha_n()$; e retornar. Caso α_j seja um terminal usar **match**(α_j).

Parsing Preditivo Recursivo

Implementação de Parser Recursivo Descendente (cont.)

- A função **parse_X** faz o seguinte:
  - Considere que existem as regras $X \rightarrow \beta_1 \mid \ldots \mid \beta_k$ para o não-terminal X.
  - Selecionar a regra $X \rightarrow \beta_i$ tal que o ***lookahead*** $\in$ **FIRST**(β_i).
    - Deve-se garantir que **FIRST**$(\beta_i) \cap$ **FIRST**$(\beta_j) = \emptyset$, $\forall i \neq j$.
    - Caso não exista tal regra, mas exista a regra $X \rightarrow \lambda$, basta retornar.
    - Caso contrário, a função falha com um erro de análise.
  - Suponha que $\beta_i = \alpha_1 \ldots \alpha_n$. Chamar **parse**_$\alpha_1()$; ... **parse**_$\alpha_n()$; e retornar. Caso α_j seja um terminal usar **match**(α_j).

Parsing Preditivo Recursivo

Implementação de Parser Recursivo Descendente (cont.)

- A função **parse_X** faz o seguinte:
  - Considere que existem as regras $X \rightarrow \beta_1 \mid \ldots \mid \beta_k$ para o não-terminal X.
  - Selecionar a regra $X \rightarrow \beta_i$ tal que o ***lookahead*** $\in$ **FIRST**(β_i).
    - Deve-se garantir que **FIRST**$(\beta_i) \cap$ **FIRST**$(\beta_j) = \emptyset$, $\forall i \neq j$.
    - Caso não exista tal regra, mas exista a regra $X \rightarrow \lambda$, basta retornar.
    - Caso contrário, a função falha com um erro de análise.
  - Suponha que $\beta_i = \alpha_1 \ldots \alpha_n$. Chamar **parse_**$\alpha_1()$; ... **parse_**$\alpha_n()$; e retornar. Caso α_j seja um terminal usar **match**(α_j).

Parsing Preditivo Recursivo

Implementação de Parser Recursivo Descendente (cont.)

- A função **parse_X** faz o seguinte:
  - Considere que existem as regras $X \to \beta_1 \mid \ldots \mid \beta_k$ para o não-terminal X.
  - Selecionar a regra $X \to \beta_i$ tal que o ***lookahead*** $\in$ **FIRST**(β_i).
    - Deve-se garantir que **FIRST**$(\beta_i) \cap$ **FIRST**$(\beta_j) = \emptyset$, $\forall i \neq j$.
    - Caso não exista tal regra, mas exista a regra $X \to \lambda$, basta retornar.
    - Caso contrário, a função falha com um erro de análise.
  - Suponha que $\beta_i = \alpha_1 \ldots \alpha_n$. Chamar **parse_**$\alpha_1()$; ... **parse_**$\alpha_n()$; e retornar. Caso α_j seja um terminal usar **match**(α_j).

Parsing Preditivo Recursivo

Implementação de Parser Recursivo Descendente (cont.)

- A função **parse_X** faz o seguinte:
  - Considere que existem as regras $X \to \beta_1 \mid \ldots \mid \beta_k$ para o não-terminal X.
  - Selecionar a regra $X \to \beta_i$ tal que o ***lookahead*** $\in$ **FIRST**(β_i).
    - Deve-se garantir que **FIRST**$(\beta_i) \cap$ **FIRST**$(\beta_j) = \emptyset$, $\forall i \neq j$.
    - Caso não exista tal regra, mas exista a regra $X \to \lambda$, basta retornar.
    - Caso contrário, a função falha com um erro de análise.
  - Suponha que $\beta_i = \alpha_1 \ldots \alpha_n$. Chamar **parse_**$\alpha_1()$; ... **parse_**$\alpha_n()$; e retornar. Caso α_j seja um terminal usar **match**(α_j).

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Exemplo 1

Considere GLC G: $S \rightarrow xyz \mid abc$

$\textbf{FIRST}(xyz) = \{x\}$, $\textbf{FIRST}(abc) = \{a\}$

$\textbf{FIRST}(S) = \textbf{FIRST}(xyz) \cup \textbf{FIRST}(abc) = \{x, a\}$

```
parse_S() {
    if (lookahead == 'x') {
        match('x'); match('y'); match('z');    // S → xyz
    } else if (lookahead == 'a') {
        match('a'); match('b'); match('c');    // S → abc
    } else erro();
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Exemplo 1

Considere GLC G: $S \rightarrow xyz \mid abc$

FIRST$(xyz) = \{x\}$, **FIRST**$(abc) = \{a\}$

FIRST$(S) =$ **FIRST**$(xyz) \cup$ **FIRST**$(abc) = \{x, a\}$

```
parse_S() {
    if (lookahead == 'x') {
        match('x'); match('y'); match('z');     // S → xyz
    } else if (lookahead == 'a') {
        match('a'); match('b'); match('c');     // S → abc
    } else erro();
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Exemplo 1

Considere GLC G: $S \rightarrow xyz \mid abc$

FIRST$(xyz) = \{x\}$, **FIRST**$(abc) = \{a\}$

FIRST(S) = **FIRST**$(xyz) \cup$ **FIRST**$(abc) = \{x, a\}$

```
parse_S() {
    if (lookahead == 'x') {
        match('x'); match('y'); match('z');    // S → xyz
    } else if (lookahead == 'a') {
        match('a'); match('b'); match('c');    // S → abc
    } else erro();
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Exemplo 1

Considere GLC G: $S \rightarrow xyz \mid abc$

FIRST$(xyz) = \{x\}$, **FIRST**$(abc) = \{a\}$

FIRST$(S) =$ **FIRST**$(xyz) \cup$ **FIRST**$(abc) = \{x, a\}$

```
parse_S() {
    if (lookahead == 'x') {
        match('x'); match('y'); match('z');     // S → xyz
    } else if (lookahead == 'a') {
        match('a'); match('b'); match('c');     // S → abc
    } else erro();
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Exemplo 1

Considere GLC G: $S \rightarrow xyz \mid abc$

FIRST$(xyz) = \{x\}$, **FIRST**$(abc) = \{a\}$

FIRST(S) = **FIRST**$(xyz) \cup$ **FIRST**$(abc) = \{x, a\}$

```
parse_S() {
    if (lookahead == 'x') {
        match('x'); match('y'); match('z');     // S → xyz
    } else if (lookahead == 'a') {
        match('a'); match('b'); match('c');     // S → abc
    } else erro();
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Exemplo 1

Considere GLC G: $S \rightarrow xyz \mid abc$

FIRST$(xyz) = \{x\}$, **FIRST**$(abc) = \{a\}$

FIRST(S) = **FIRST**$(xyz) \cup$ **FIRST**$(abc) = \{x, a\}$

```
parse_S() {
    if (lookahead == 'x') {
        match('x'); match('y'); match('z');     // S → xyz
    } else if (lookahead == 'a') {
        match('a'); match('b'); match('c');     // S → abc
    } else erro();
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Exemplo 1

Considere GLC G: $S \rightarrow xyz \mid abc$

FIRST$(xyz) = \{x\}$, **FIRST**$(abc) = \{a\}$

FIRST(S) = **FIRST**$(xyz) \cup$ **FIRST**$(abc) = \{x, a\}$

```
parse_S() {
    if (lookahead == 'x') {
        match('x'); match('y'); match('z');     // S → xyz
    } else if (lookahead == 'a') {
        match('a'); match('b'); match('c');     // S → abc
    } else erro();
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Exemplo 2

Considere GLC G: $S \to A \mid B$
$A \to x \mid y$
$B \to z$

$\textbf{FIRST}(x) = \{x\}$, $\textbf{FIRST}(y) = \{y\}$, $\textbf{FIRST}(A) = \{x, y\}$

$\textbf{FIRST}(z) = \{z\}$, $\textbf{FIRST}(B) = \{z\}$

$\textbf{FIRST}(S) = \textbf{FIRST}(A) \cup \textbf{FIRST}(B) = \{x, y, z\}$

```
parse_A() {
    if (lookahead == 'x') {
        match('x');    // A → x
    } else if (lookahead == 'y') {
        match('y');    // A → y
    } else erro();
}
```

```
parse_B() {
    if (lookahead == 'z') {
        match('z');    // B → z
    } else erro();
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Exemplo 2

Considere GLC G: $S \to A \mid B$
$A \to x \mid y$
$B \to z$

$\textbf{FIRST}(x) = \{x\}$, $\textbf{FIRST}(y) = \{y\}$, $\textbf{FIRST}(A) = \{x, y\}$

$\textbf{FIRST}(z) = \{z\}$, $\textbf{FIRST}(B) = \{z\}$

$\textbf{FIRST}(S) = \textbf{FIRST}(A) \cup \textbf{FIRST}(B) = \{x, y, z\}$

```
parse_A() {
    if (lookahead == 'x') {
        match('x');    // A → x
    } else if (lookahead == 'y') {
        match('y');    // A → y
    } else erro();
}
```

```
parse_B() {
    if (lookahead == 'z') {
        match('z');    // B → z
    } else erro();
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Exemplo 2

Considere GLC G: $S \to A \mid B$
$A \to x \mid y$
$B \to z$

$\textbf{FIRST}(x) = \{x\}$, $\textbf{FIRST}(y) = \{y\}$, $\textbf{FIRST}(A) = \{x, y\}$

$\textbf{FIRST}(z) = \{z\}$, $\textbf{FIRST}(B) = \{z\}$

$\textbf{FIRST}(S) = \textbf{FIRST}(A) \cup \textbf{FIRST}(B) = \{x, y, z\}$

```
parse_A() {
    if (lookahead == 'x') {
        match('x');    // A → x
    } else if (lookahead == 'y') {
        match('y');    // A → y
    } else erro();
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Exemplo 2

Considere GLC G: $S \to A \mid B$
$A \to x \mid y$
$B \to z$

$\textbf{FIRST}(x) = \{x\}$, $\textbf{FIRST}(y) = \{y\}$, $\textbf{FIRST}(A) = \{x, y\}$

$\textbf{FIRST}(z) = \{z\}$, $\textbf{FIRST}(B) = \{z\}$

$\textbf{FIRST}(S) = \textbf{FIRST}(A) \cup \textbf{FIRST}(B) = \{x, y, z\}$

```
parse_A() {
    if (lookahead == 'x') {
        match('x');    // A → x
    } else if (lookahead == 'y') {
        match('y');    // A → y
    } else erro();
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Exemplo 2

Considere GLC G: $S \to A \mid B$
$A \to x \mid y$
$B \to z$

$\textbf{FIRST}(x) = \{x\}$, $\textbf{FIRST}(y) = \{y\}$, $\textbf{FIRST}(A) = \{x, y\}$

$\textbf{FIRST}(z) = \{z\}$, $\textbf{FIRST}(B) = \{z\}$

$\textbf{FIRST}(S) = \textbf{FIRST}(A) \cup \textbf{FIRST}(B) = \{x, y, z\}$

```
parse_A() {
    if (lookahead == 'x') {
        match('x');    // A → x
    } else if (lookahead == 'y') {
        match('y');    // A → y
    } else erro();
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Exemplo 2

Considere GLC G: $S \to A \mid B$
$A \to x \mid y$
$B \to z$

$\textbf{FIRST}(x) = \{x\}$, $\textbf{FIRST}(y) = \{y\}$, $\textbf{FIRST}(A) = \{x, y\}$

$\textbf{FIRST}(z) = \{z\}$, $\textbf{FIRST}(B) = \{z\}$

$\textbf{FIRST}(S) = \textbf{FIRST}(A) \cup \textbf{FIRST}(B) = \{x, y, z\}$

```
parse_A() {
    if (lookahead == 'x') {
        match('x');    // A → x
    } else if (lookahead == 'y') {
        match('y');    // A → y
    } else erro();
}
```

```
parse_B() {
    if (lookahead == 'z') {
        match('z');    // B → z
    } else erro();
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Exemplo 2

Considere GLC G: $S \to A \mid B$
$A \to x \mid y$
$B \to z$

$\textbf{FIRST}(x) = \{x\}$, $\textbf{FIRST}(y) = \{y\}$, $\textbf{FIRST}(A) = \{x, y\}$

$\textbf{FIRST}(z) = \{z\}$, $\textbf{FIRST}(B) = \{z\}$

$\textbf{FIRST}(S) = \textbf{FIRST}(A) \cup \textbf{FIRST}(B) = \{x, y, z\}$

```
parse_A() {
    if (lookahead == 'x') {
        match('x');    // A → x
    } else if (lookahead == 'y') {
        match('y');    // A → y
    } else erro();
}
```

```
parse_B() {
    if (lookahead == 'z') {
        match('z');    // B → z
    } else erro();
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Exemplo 2

Considere GLC G: $S \to A \mid B$
$A \to x \mid y$
$B \to z$

$\textbf{FIRST}(x) = \{x\}$, $\textbf{FIRST}(y) = \{y\}$, $\textbf{FIRST}(A) = \{x, y\}$

$\textbf{FIRST}(z) = \{z\}$, $\textbf{FIRST}(B) = \{z\}$

$\textbf{FIRST}(S) = \textbf{FIRST}(A) \cup \textbf{FIRST}(B) = \{x, y, z\}$

```
parse_A() {
    if (lookahead == 'x') {
        match('x');   // A → x
    } else if (lookahead == 'y') {
        match('y');   // A → y
    } else erro();
}
```

```
parse_B() {
    if (lookahead == 'z') {
        match('z');   // B → z
    } else erro();
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Exemplo 2

Considere GLC G: $S \to A \mid B$
$A \to x \mid y$
$B \to z$

$\textbf{FIRST}(x) = \{x\}$, $\textbf{FIRST}(y) = \{y\}$, $\textbf{FIRST}(A) = \{x, y\}$

$\textbf{FIRST}(z) = \{z\}$, $\textbf{FIRST}(B) = \{z\}$

$\textbf{FIRST}(S) = \textbf{FIRST}(A) \cup \textbf{FIRST}(B) = \{x, y, z\}$

```
parse_A() {
    if (lookahead == 'x') {
        match('x');   // A → x
    } else if (lookahead == 'y') {
        match('y');   // A → y
    } else erro();
}
```

```
parse_B() {
    if (lookahead == 'z') {
        match('z');   // B → z
    } else erro();
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Exemplo 2

Considere GLC G: $S \to A \mid B$
$A \to x \mid y$
$B \to z$

$\textbf{FIRST}(x) = \{x\}$, $\textbf{FIRST}(y) = \{y\}$, $\textbf{FIRST}(A) = \{x, y\}$

$\textbf{FIRST}(z) = \{z\}$, $\textbf{FIRST}(B) = \{z\}$

$\textbf{FIRST}(S) = \textbf{FIRST}(A) \cup \textbf{FIRST}(B) = \{x, y, z\}$

```
parse_A() {
    if (lookahead == 'x') {
        match('x');   // A → x
    } else if (lookahead == 'y') {
        match('y');   // A → y
    } else erro();
}
```

```
parse_B() {
    if (lookahead == 'z') {
        match('z');   // B → z
    } else erro();
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Exemplo 2

Considere GLC G: $S \to A \mid B$
$A \to x \mid y$
$B \to z$

$\textbf{FIRST}(x) = \{x\}$, $\textbf{FIRST}(y) = \{y\}$, $\textbf{FIRST}(A) = \{x, y\}$

$\textbf{FIRST}(z) = \{z\}$, $\textbf{FIRST}(B) = \{z\}$

$\textbf{FIRST}(S) = \textbf{FIRST}(A) \cup \textbf{FIRST}(B) = \{x, y, z\}$

```
parse_A() {
    if (lookahead == 'x') {
        match('x');   // A → x
    } else if (lookahead == 'y') {
        match('y');   // A → y
    } else erro();
}
```

```
parse_B() {
    if (lookahead == 'z') {
        match('z');   // B → z
    } else erro();
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Exemplo 2 (cont.)

Considere GLC G: $S \to A \mid B$
$A \to x \mid y$
$B \to z$

$\textbf{FIRST}(x) = \{x\}$, $\textbf{FIRST}(y) = \{y\}$, $\textbf{FIRST}(A) = \{x, y\}$

$\textbf{FIRST}(z) = \{z\}$, $\textbf{FIRST}(B) = \{z\}$

$\textbf{FIRST}(S) = \textbf{FIRST}(A) \cup \textbf{FIRST}(B) = \{x, y, z\}$

```
parse_S() {
    if (lookahead == 'x') || (lookahead == 'y') {
        parse_A();                          // S → A
    } else if (lookahead == 'z') {
        parse_B();                          // S → B
    } else erro();
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Exemplo 2 (cont.)

Considere GLC G: $S \to A \mid B$
$A \to x \mid y$
$B \to z$

$\textbf{FIRST}(x) = \{x\}$, $\textbf{FIRST}(y) = \{y\}$, $\textbf{FIRST}(A) = \{x, y\}$

$\textbf{FIRST}(z) = \{z\}$, $\textbf{FIRST}(B) = \{z\}$

$\textbf{FIRST}(S) = \textbf{FIRST}(A) \cup \textbf{FIRST}(B) = \{x, y, z\}$

```
parse_S() {
    if (lookahead == 'x') || (lookahead == 'y') {
        parse_A();                    // S → A
    } else if (lookahead == 'z') {
        parse_B();                    // S → B
    } else erro();
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Exemplo 2 (cont.)

Considere GLC G: $S \to A \mid B$
$A \to x \mid y$
$B \to z$

$\textbf{FIRST}(x) = \{x\}$, $\textbf{FIRST}(y) = \{y\}$, $\textbf{FIRST}(A) = \{x, y\}$

$\textbf{FIRST}(z) = \{z\}$, $\textbf{FIRST}(B) = \{z\}$

$\textbf{FIRST}(S) = \textbf{FIRST}(A) \cup \textbf{FIRST}(B) = \{x, y, z\}$

```
parse_S() {
    if (lookahead == 'x') || (lookahead == 'y') {
        parse_A();                          // S → A
    } else if (lookahead == 'z') {
        parse_B();                          // S → B
    } else erro();
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Exemplo 2 (cont.)

Considere GLC G: $S \to A \mid B$
$A \to x \mid y$
$B \to z$

$\mathbf{FIRST}(x) = \{x\}$, $\mathbf{FIRST}(y) = \{y\}$, $\mathbf{FIRST}(A) = \{x, y\}$

$\mathbf{FIRST}(z) = \{z\}$, $\mathbf{FIRST}(B) = \{z\}$

$\mathbf{FIRST}(S) = \mathbf{FIRST}(A) \cup \mathbf{FIRST}(B) = \{x, y, z\}$

```
parse_S() {
    if (lookahead == 'x') || (lookahead == 'y') {
        parse_A();                          // S → A
    } else if (lookahead == 'z') {
        parse_B();                          // S → B
    } else erro();
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Exemplo 2 (cont.)

Considere GLC G: $S \to A \mid B$
$A \to x \mid y$
$B \to z$

$\textbf{FIRST}(x) = \{x\}$, $\textbf{FIRST}(y) = \{y\}$, $\textbf{FIRST}(A) = \{x, y\}$

$\textbf{FIRST}(z) = \{z\}$, $\textbf{FIRST}(B) = \{z\}$

$\textbf{FIRST}(S) = \textbf{FIRST}(A) \cup \textbf{FIRST}(B) = \{x, y, z\}$

```
parse_S() {
    if (lookahead == 'x') || (lookahead == 'y') {
        parse_A();                          // S → A
    } else if (lookahead == 'z') {
        parse_B();                          // S → B
    } else erro();
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Exemplo 2 (cont.)

Considere GLC G: $S \to A \mid B$
$A \to x \mid y$
$B \to z$

$\textbf{FIRST}(x) = \{x\}$, $\textbf{FIRST}(y) = \{y\}$, $\textbf{FIRST}(A) = \{x, y\}$

$\textbf{FIRST}(z) = \{z\}$, $\textbf{FIRST}(B) = \{z\}$

$\textbf{FIRST}(S) = \textbf{FIRST}(A) \cup \textbf{FIRST}(B) = \{x, y, z\}$

```
parse_S() {
    if (lookahead == 'x') || (lookahead == 'y') {
        parse_A();                          // S → A
    } else if (lookahead == 'z') {
        parse_B();                          // S → B
    } else erro();
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Exemplo 2 (cont.)

Considere GLC G: $S \to A \mid B$
$A \to x \mid y$
$B \to z$

$\textbf{FIRST}(x) = \{x\}$, $\textbf{FIRST}(y) = \{y\}$, $\textbf{FIRST}(A) = \{x, y\}$

$\textbf{FIRST}(z) = \{z\}$, $\textbf{FIRST}(B) = \{z\}$

$\textbf{FIRST}(S) = \textbf{FIRST}(A) \cup \textbf{FIRST}(B) = \{x, y, z\}$

```
parse_S() {
    if (lookahead == 'x') || (lookahead == 'y') {
        parse_A();                          // S → A
    } else if (lookahead == 'z') {
        parse_B();                          // S → B
    } else erro();
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Exemplo 3

Considere GLC G: $E \rightarrow \mathbf{id} = \mathbf{n} \mid \{\ L\ \}$

$L \rightarrow\ E\ ;\ L \mid \lambda$

$\mathbf{FIRST}(E) = \mathbf{FIRST}(\mathbf{id} = \mathbf{n}) \cup \mathbf{FIRST}(\{\ L\ \}) = \{\mathbf{id},\ '\{'\}$

$\mathbf{FIRST}(L) = \mathbf{FIRST}(E\ ;\ L) \cup \mathbf{FIRST}(\lambda) = \{\mathbf{id},\ '\{', \lambda\}$

```
parse_E() {
    if (lookahead == id) {
        match(id);
        match('=');   // E → id = n
        match(n);
    } else if (lookahead == '{') {
        match('{');
        parse_L();    // E → { L }
        match('}');
    } else erro();
}
```

```
parse_L() {
    if (lookahead == id) ||
    (lookahead == '{') {
        parse_E();
        match(';');   // L → E ; L
        parse_L();
    } else ;          // L → λ
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Exemplo 3

Considere GLC G: $E \rightarrow \mathbf{id} = \mathbf{n} \mid \{\ L\ \}$

$L \rightarrow\ E\ ;\ L \mid \lambda$

$\mathbf{FIRST}(E) = \mathbf{FIRST}(\mathbf{id} = \mathbf{n}) \cup \mathbf{FIRST}(\{\ L\ \}) = \{\mathbf{id},\ '\{'\}$

$\mathbf{FIRST}(L) = \mathbf{FIRST}(E\ ;\ L) \cup \mathbf{FIRST}(\lambda) = \{\mathbf{id},\ '\{', \lambda\}$

```
parse_E() {
    if (lookahead == id) {
        match(id);
        match('=');   // E → id = n
        match(n);
    } else if (lookahead == '{') {
        match('{');
        parse_L();    // E → { L }
        match('}');
    } else erro();
}
```

```
parse_L() {
    if (lookahead == id) ||
    (lookahead == '{') {
        parse_E();
        match(';');   // L → E ; L
        parse_L();
    } else ;          // L → λ
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Exemplo 3

Considere GLC G: $E \rightarrow \mathbf{id} = \mathbf{n} \mid \{\ L\ \}$

$L \rightarrow\ E\ ;\ L \mid \lambda$

$\mathbf{FIRST}(E) = \mathbf{FIRST}(\mathbf{id} = \mathbf{n}) \cup \mathbf{FIRST}(\{\ L\ \}) = \{\mathbf{id},\ '\{'\}$

$\mathbf{FIRST}(L) = \mathbf{FIRST}(E\ ;\ L) \cup \mathbf{FIRST}(\lambda) = \{\mathbf{id},\ '\{', \lambda\}$

```
parse_E() {
    if (lookahead == id) {
        match(id);
        match('=');   // E → id = n
        match(n);
    } else if (lookahead == '{') {
        match('{');
        parse_L();    // E → { L }
        match('}');
    } else erro();
}
```

```
parse_L() {
    if (lookahead == id) ||
    (lookahead == '{') {
        parse_E();
        match(';');   // L → E ; L
        parse_L();
    } else ;          // L → λ
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Exemplo 3

Considere GLC G: $E \rightarrow \mathbf{id} = \mathbf{n} \mid \{\ L\ \}$

$L \rightarrow\ E\ ;\ L \mid \lambda$

$\mathbf{FIRST}(E) = \mathbf{FIRST}(\mathbf{id} = \mathbf{n}) \cup \mathbf{FIRST}(\{\ L\ \}) = \{\mathbf{id},\ '\{'\}$

$\mathbf{FIRST}(L) = \mathbf{FIRST}(E\ ;\ L) \cup \mathbf{FIRST}(\lambda) = \{\mathbf{id},\ '\{', \lambda\}$

```
parse_E() {
    if (lookahead == id) {
        match(id);
        match('=');   // E → id = n
        match(n);
    } else if (lookahead == '{') {
        match('{');
        parse_L();    // E → { L }
        match('}');
    } else erro();
}
```

```
parse_L() {
    if (lookahead == id) ||
    (lookahead == '{') {
        parse_E();
        match(';');   // L → E ; L
        parse_L();
    } else ;          // L → λ
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Exemplo 3

Considere GLC G: $E \rightarrow \mathbf{id} = \mathbf{n} \mid \{\ L\ \}$

$L \rightarrow\ E\ ;\ L \mid \lambda$

$\mathbf{FIRST}(E) = \mathbf{FIRST}(\mathbf{id} = \mathbf{n}) \cup \mathbf{FIRST}(\{\ L\ \}) = \{\mathbf{id},\ '\{'\}$

$\mathbf{FIRST}(L) = \mathbf{FIRST}(E\ ;\ L) \cup \mathbf{FIRST}(\lambda) = \{\mathbf{id},\ '\{', \lambda\}$

```
parse_E() {
    if (lookahead == id) {
        match(id);
        match('=');   // E → id = n
        match(n);
    } else if (lookahead == '{') {
        match('{');
        parse_L();    // E → { L }
        match('}');
    } else erro();
}
```

```
parse_L() {
    if (lookahead == id) ||
    (lookahead == '{') {
        parse_E();
        match(';');   // L → E ; L
        parse_L();
    } else ;          // L → λ
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Exemplo 3

Considere GLC G: $E \rightarrow \mathbf{id} = \mathbf{n} \mid \{\ L\ \}$

$L \rightarrow\ E\ ;\ L \mid \lambda$

$\mathbf{FIRST}(E) = \mathbf{FIRST}(\mathbf{id} = \mathbf{n}) \cup \mathbf{FIRST}(\{\ L\ \}) = \{\mathbf{id},\ '\{'\}$

$\mathbf{FIRST}(L) = \mathbf{FIRST}(E\ ;\ L) \cup \mathbf{FIRST}(\lambda) = \{\mathbf{id},\ '\{', \lambda\}$

```
parse_E() {
    if (lookahead == id) {
        match(id);
        match('=');   // E → id = n
        match(n);
    } else if (lookahead == '{') {
        match('{');
        parse_L();    // E → { L }
        match('}');
    } else erro();
}
```

```
parse_L() {
    if (lookahead == id) ||
    (lookahead == '{') {
        parse_E();
        match(';');   // L → E ; L
        parse_L();
    } else ;          // L → λ
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Exemplo 3

Considere GLC G: $E \rightarrow \mathbf{id} = \mathbf{n} \mid \{\ L\ \}$

$L \rightarrow\ E\ ;\ L \mid \lambda$

$\mathbf{FIRST}(E) = \mathbf{FIRST}(\mathbf{id} = \mathbf{n}) \cup \mathbf{FIRST}(\{\ L\ \}) = \{\mathbf{id},\ '\{'\}$

$\mathbf{FIRST}(L) = \mathbf{FIRST}(E\ ;\ L) \cup \mathbf{FIRST}(\lambda) = \{\mathbf{id},\ '\{', \lambda\}$

```
parse_E() {
    if (lookahead == id) {
        match(id);
        match('=');   // E → id = n
        match(n);
    } else if (lookahead == '{') {
        match('{');
        parse_L();    // E → { L }
        match('}');
    } else erro();
}
```

```
parse_L() {
    if (lookahead == id) ||
    (lookahead == '{') {
        parse_E();
        match(';');   // L → E ; L
        parse_L();
    } else ;          // L → λ
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Exemplo 3

Considere GLC G: $E \rightarrow \mathbf{id} = \mathbf{n} \mid \{\ L\ \}$

$L \rightarrow\ E\ ;\ L \mid \lambda$

$\mathbf{FIRST}(E) = \mathbf{FIRST}(\mathbf{id} = \mathbf{n}) \cup \mathbf{FIRST}(\{\ L\ \}) = \{\mathbf{id},\ '\{'\}$

$\mathbf{FIRST}(L) = \mathbf{FIRST}(E\ ;\ L) \cup \mathbf{FIRST}(\lambda) = \{\mathbf{id},\ '\{', \lambda\}$

```
parse_E() {
    if (lookahead == id) {
        match(id);
        match('=');   // E → id = n
        match(n);
    } else if (lookahead == '{') {
        match('{');
        parse_L();    // E → { L }
        match('}');
    } else erro();
}
```

```
parse_L() {
    if (lookahead == id) ||
    (lookahead == '{') {
        parse_E();
        match(';');   // L → E ; L
        parse_L();
    } else ;          // L → λ
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Exemplo 3

Considere GLC G: $E \rightarrow \mathbf{id} = \mathbf{n} \mid \{\ L\ \}$

$L \rightarrow\ E\ ;\ L \mid \lambda$

$\mathbf{FIRST}(E) = \mathbf{FIRST}(\mathbf{id} = \mathbf{n}) \cup \mathbf{FIRST}(\{\ L\ \}) = \{\mathbf{id},\ '\{'\}$

$\mathbf{FIRST}(L) = \mathbf{FIRST}(E\ ;\ L) \cup \mathbf{FIRST}(\lambda) = \{\mathbf{id},\ '\{', \lambda\}$

```
parse_E() {
    if (lookahead == id) {
        match(id);
        match('=');   // E → id = n
        match(n);
    } else if (lookahead == '{') {
        match('{');
        parse_L();    // E → { L }
        match('}');
    } else erro();
}
```

```
parse_L() {
    if (lookahead == id) ||
      (lookahead == '{') {
        parse_E();
        match(';');   // L → E ; L
        parse_L();
    } else ;          // L → λ
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Exemplo 3

Considere GLC G: $E \rightarrow \mathbf{id} = \mathbf{n} \mid \{\ L\ \}$

$L \rightarrow\ E\ ;\ L \mid \lambda$

$\mathbf{FIRST}(E) = \mathbf{FIRST}(\mathbf{id} = \mathbf{n}) \cup \mathbf{FIRST}(\{\ L\ \}) = \{\mathbf{id},\ '\{'\}$

$\mathbf{FIRST}(L) = \mathbf{FIRST}(E\ ;\ L) \cup \mathbf{FIRST}(\lambda) = \{\mathbf{id},\ '\{', \lambda\}$

```
parse_E() {
    if (lookahead == id) {
        match(id);
        match('=');   // E → id = n
        match(n);
    } else if (lookahead == '{') {
        match('{');
        parse_L();    // E → { L }
        match('}');
    } else erro();
}
```

```
parse_L() {
    if (lookahead == id) ||
      (lookahead == '{') {
        parse_E();
        match(';');   // L → E ; L
        parse_L();
    } else ;          // L → λ
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Exemplo 3

Considere GLC G: $E \rightarrow \mathbf{id} = \mathbf{n} \mid \{\ L\ \}$

$L \rightarrow\ E\ ;\ L \mid \lambda$

$\mathbf{FIRST}(E) = \mathbf{FIRST}(\mathbf{id} = \mathbf{n}) \cup \mathbf{FIRST}(\{\ L\ \}) = \{\mathbf{id},\ '\{'\}$

$\mathbf{FIRST}(L) = \mathbf{FIRST}(E\ ;\ L) \cup \mathbf{FIRST}(\lambda) = \{\mathbf{id},\ '\{', \lambda\}$

```
parse_E() {
    if (lookahead == id) {
        match(id);
        match('=');   // E → id = n
        match(n);
    } else if (lookahead == '{') {
        match('{');
        parse_L();    // E → { L }
        match('}');
    } else erro();
}
```

```
parse_L() {
    if (lookahead == id) ||
      (lookahead == '{') {
        parse_E();
        match(';');   // L → E ; L
        parse_L();
    } else ;          // L → λ
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Exemplo 4

Considere GLC G: $E \rightarrow \mathbf{id} = \mathbf{n} \mid \{\ L\ \} \mid \lambda$

$L \rightarrow\ E\ ;\ L$

$\mathbf{FIRST}(E) = \mathbf{FIRST}(\mathbf{id} = \mathbf{n}) \cup \mathbf{FIRST}(\{\ L\ \}) \cup \mathbf{FIRST}(\lambda) = \{\mathbf{id},\ '\{', \lambda\}$

$\mathbf{FIRST}(L) = \mathbf{FIRST}(E\ ;\ L) = \{\mathbf{id},\ '\{',\ ';'\}$

```
parse_E() {
    if (lookahead == id) {
        match(id);
        match('=');    // E → id = n
        match(n);
    } else if (lookahead == '{') {
        match('{');
        parse_L();     // E → { L }
        match('}');
    } else ;           // E → λ
}
```

```
parse_L() {
    if (lookahead == id) ||
    (lookahead == '{') ||
    (lookahead == ';') {
        parse_E();
        match(';');    // L → E ; L
        parse_L();
    } else erro();
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Exemplo 4

Considere GLC G: $E \rightarrow \mathbf{id} = \mathbf{n} \mid \{\ L\ \} \mid \lambda$

$L \rightarrow\ E\ ;\ L$

$\mathbf{FIRST}(E) = \mathbf{FIRST}(\mathbf{id} = \mathbf{n}) \cup \mathbf{FIRST}(\{\ L\ \}) \cup \mathbf{FIRST}(\lambda) = \{\mathbf{id},\ '\{', \lambda\}$

$\mathbf{FIRST}(L) = \mathbf{FIRST}(E\ ;\ L) = \{\mathbf{id},\ '\{',\ ';'\}$

```
parse_E() {
    if (lookahead == id) {
        match(id);
        match('=');   // E → id = n
        match(n);
    } else if (lookahead == '{') {
        match('{');
        parse_L();    // E → { L }
        match('}');
    } else ;          // E → λ
}
```

```
parse_L() {
    if (lookahead == id) ||
    (lookahead == '{') ||
    (lookahead == ';') {
        parse_E();
        match(';');   // L → E ; L
        parse_L();
    } else erro();
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Exemplo 4

Considere GLC G: $E \rightarrow \mathbf{id} = \mathbf{n} \mid \{\ L\ \} \mid \lambda$

$L \rightarrow E\ ;\ L$

$\mathbf{FIRST}(E) = \mathbf{FIRST}(\mathbf{id} = \mathbf{n}) \cup \mathbf{FIRST}(\{\ L\ \}) \cup \mathbf{FIRST}(\lambda) = \{\mathbf{id},\ '\{', \lambda\}$

$\mathbf{FIRST}(L) = \mathbf{FIRST}(E\ ;\ L) = \{\mathbf{id},\ '\{',\ ';'\}$

```
parse_E() {
    if (lookahead == id) {
        match(id);
        match('=');   // E → id = n
        match(n);
    } else if (lookahead == '{') {
        match('{');
        parse_L();    // E → { L }
        match('}');
    } else ;          // E → λ
}
```

```
parse_L() {
    if (lookahead == id) ||
       (lookahead == '{') ||
       (lookahead == ';') {
        parse_E();
        match(';');   // L → E ; L
        parse_L();
    } else erro();
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Exemplo 4

Considere GLC G: $E \rightarrow \mathbf{id} = \mathbf{n} \mid \{\ L\ \} \mid \lambda$

$L \rightarrow E\ ;\ L$

$\mathbf{FIRST}(E) = \mathbf{FIRST}(\mathbf{id} = \mathbf{n}) \cup \mathbf{FIRST}(\{\ L\ \}) \cup \mathbf{FIRST}(\lambda) = \{\mathbf{id},\ '\{', \lambda\}$

$\mathbf{FIRST}(L) = \mathbf{FIRST}(E\ ;\ L) = \{\mathbf{id},\ '\{',\ ';'\}$

```
parse_E() {
    if (lookahead == id) {
        match(id);
        match('=');   // E → id = n
        match(n);
    } else if (lookahead == '{') {
        match('{');
        parse_L();    // E → { L }
        match('}');
    } else ;          // E → λ
}
```

```
parse_L() {
    if (lookahead == id) ||
    (lookahead == '{') ||
    (lookahead == ';') {
        parse_E();
        match(';');   // L → E ; L
        parse_L();
    } else erro();
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Exemplo 4

Considere GLC G: $E \rightarrow \mathbf{id} = \mathbf{n} \mid \{\ L\ \} \mid \lambda$

$L \rightarrow E\ ;\ L$

$\mathbf{FIRST}(E) = \mathbf{FIRST}(\mathbf{id} = \mathbf{n}) \cup \mathbf{FIRST}(\{\ L\ \}) \cup \mathbf{FIRST}(\lambda) = \{\mathbf{id},\ '\{', \lambda\}$

$\mathbf{FIRST}(L) = \mathbf{FIRST}(E\ ;\ L) = \{\mathbf{id},\ '\{',\ ';'\}$

```
parse_E() {
    if (lookahead == id) {
        match(id);
        match('=');   // E → id = n
        match(n);
    } else if (lookahead == '{') {
        match('{');
        parse_L();    // E → { L }
        match('}');
    } else ;          // E → λ
}
```

```
parse_L() {
    if (lookahead == id) ||
    (lookahead == '{') ||
    (lookahead == ';') {
        parse_E();
        match(';');   // L → E ; L
        parse_L();
    } else erro();
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Exemplo 4

Considere GLC G: $E \rightarrow \mathbf{id} = \mathbf{n} \mid \{\ L\ \} \mid \lambda$

$L \rightarrow E\ ;\ L$

$\mathbf{FIRST}(E) = \mathbf{FIRST}(\mathbf{id} = \mathbf{n}) \cup \mathbf{FIRST}(\{\ L\ \}) \cup \mathbf{FIRST}(\lambda) = \{\mathbf{id},\ '\{', \lambda\}$

$\mathbf{FIRST}(L) = \mathbf{FIRST}(E\ ;\ L) = \{\mathbf{id},\ '\{',\ ';'\}$

```
parse_E() {
    if (lookahead == id) {
        match(id);
        match('=');   // E → id = n
        match(n);
    } else if (lookahead == '{') {
        match('{');
        parse_L();    // E → { L }
        match('}');
    } else ;          // E → λ
}
```

```
parse_L() {
    if (lookahead == id) ||
    (lookahead == '{') ||
    (lookahead == ';') {
        parse_E();
        match(';');   // L → E ; L
        parse_L();
    } else erro();
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Exemplo 4

Considere GLC G: $E \rightarrow \mathbf{id} = \mathbf{n} \mid \{\ L\ \} \mid \lambda$

$L \rightarrow\ E\ ;\ L$

$\mathbf{FIRST}(E) = \mathbf{FIRST}(\mathbf{id} = \mathbf{n}) \cup \mathbf{FIRST}(\{\ L\ \}) \cup \mathbf{FIRST}(\lambda) = \{\mathbf{id},\ '\{', \lambda\}$

$\mathbf{FIRST}(L) = \mathbf{FIRST}(E\ ;\ L) = \{\mathbf{id},\ '\{',\ ';'\}$

```
parse_E() {
    if (lookahead == id) {
        match(id);
        match('=');   // E → id = n
        match(n);
    } else if (lookahead == '{') {
        match('{');
        parse_L();    // E → { L }
        match('}');
    } else ;          // E → λ
}
```

```
parse_L() {
    if (lookahead == id) ||
       (lookahead == '{') ||
       (lookahead == ';') {
        parse_E();
        match(';');   // L → E ; L
        parse_L();
    } else erro();
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Exemplo 4

Considere GLC G: $E \rightarrow \mathbf{id} = \mathbf{n} \mid \{\ L\ \} \mid \lambda$

$L \rightarrow E\ ;\ L$

$\mathbf{FIRST}(E) = \mathbf{FIRST}(\mathbf{id} = \mathbf{n}) \cup \mathbf{FIRST}(\{\ L\ \}) \cup \mathbf{FIRST}(\lambda) = \{\mathbf{id},\ '\{', \lambda\}$

$\mathbf{FIRST}(L) = \mathbf{FIRST}(E\ ;\ L) = \{\mathbf{id},\ '\{',\ ';'\}$

```
parse_E() {
    if (lookahead == id) {
        match(id);
        match('=');    // E → id = n
        match(n);
    } else if (lookahead == '{') {
        match('{');
        parse_L();     // E → { L }
        match('}');
    } else ;           // E → λ
}
```

```
parse_L() {
    if (lookahead == id) ||
       (lookahead == '{') ||
       (lookahead == ';') {
        parse_E();
        match(';');    // L → E ; L
        parse_L();
    } else erro();
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Exemplo 4

Considere GLC G: $E \rightarrow \mathbf{id} = \mathbf{n} \mid \{\ L\ \} \mid \lambda$

$L \rightarrow E\ ;\ L$

$\mathbf{FIRST}(E) = \mathbf{FIRST}(\mathbf{id} = \mathbf{n}) \cup \mathbf{FIRST}(\{\ L\ \}) \cup \mathbf{FIRST}(\lambda) = \{\mathbf{id},\ '\{', \lambda\}$

$\mathbf{FIRST}(L) = \mathbf{FIRST}(E\ ;\ L) = \{\mathbf{id},\ '\{',\ ';'\}$

```
parse_E() {
    if (lookahead == id) {
        match(id);
        match('=');    // E → id = n
        match(n);
    } else if (lookahead == '{') {
        match('{');
        parse_L();     // E → { L }
        match('}');
    } else ;           // E → λ
}
```

```
parse_L() {
    if (lookahead == id) ||
      (lookahead == '{') ||
      (lookahead == ';') {
        parse_E();
        match(';');    // L → E ; L
        parse_L();
    } else erro();
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Exemplo 4

Considere GLC G: $E \rightarrow \mathbf{id} = \mathbf{n} \mid \{\ L\ \} \mid \lambda$

$L \rightarrow E\ ;\ L$

$\mathbf{FIRST}(E) = \mathbf{FIRST}(\mathbf{id} = \mathbf{n}) \cup \mathbf{FIRST}(\{\ L\ \}) \cup \mathbf{FIRST}(\lambda) = \{\mathbf{id},\ '\{', \lambda\}$

$\mathbf{FIRST}(L) = \mathbf{FIRST}(E\ ;\ L) = \{\mathbf{id},\ '\{',\ ';'\}$

```
parse_E() {
    if (lookahead == id) {
        match(id);
        match('=');   // E → id = n
        match(n);
    } else if (lookahead == '{') {
        match('{');
        parse_L();    // E → { L }
        match('}');
    } else ;          // E → λ
}
```

```
parse_L() {
    if (lookahead == id) ||
      (lookahead == '{') ||
      (lookahead == ';') {
        parse_E();
        match(';');   // L → E ; L
        parse_L();
    } else erro();
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Exemplo 4

Considere GLC G: $E \rightarrow \mathbf{id} = \mathbf{n} \mid \{\ L\ \} \mid \lambda$

$L \rightarrow E\ ;\ L$

$\mathbf{FIRST}(E) = \mathbf{FIRST}(\mathbf{id} = \mathbf{n}) \cup \mathbf{FIRST}(\{\ L\ \}) \cup \mathbf{FIRST}(\lambda) = \{\mathbf{id},\ '\{', \lambda\}$

$\mathbf{FIRST}(L) = \mathbf{FIRST}(E\ ;\ L) = \{\mathbf{id},\ '\{',\ ';'\}$

```
parse_E() {
    if (lookahead == id) {
        match(id);
        match('=');   // E → id = n
        match(n);
    } else if (lookahead == '{') {
        match('{');
        parse_L();    // E → { L }
        match('}');
    } else ;          // E → λ
}
```

```
parse_L() {
    if (lookahead == id) ||
      (lookahead == '{') ||
      (lookahead == ';') {
        parse_E();
        match(';');   // L → E ; L
        parse_L();
    } else erro();
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Problema 1

Considere GLC G: $S \rightarrow ab \mid ac$

$\textbf{FIRST}(ab) = \{a\}$, $\textbf{FIRST}(ac) = \{a\}$

$\textbf{FIRST}(ab) \cap \textbf{FIRST}(ac) = \{a\} \neq \emptyset$

```
parse_S() {
  if (lookahead == 'a') {
    match('a'); match('b');   // S --> ab
  } else if (lookahead == 'a') {
    match('a'); match('c');   // S --> ac
  } else error();
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Problema 1

Considere GLC G: $S \rightarrow ab \mid ac$

FIRST$(ab) = \{a\}$, **FIRST**$(ac) = \{a\}$

FIRST$(ab) \cap$ **FIRST**$(ac) = \{a\} \neq \emptyset$

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Problema 1

Considere GLC G: $S \rightarrow ab \mid ac$

FIRST$(ab) = \{a\}$, **FIRST**$(ac) = \{a\}$

FIRST$(ab) \cap$ **FIRST**$(ac) = \{a\} \neq \emptyset$

```
parse_S() {
    if (lookahead == 'a') {
        match('a'); match('b');          // S → ab
    } else if (lookahead == 'a') {
        match('a'); match('c');          // S → ac
    } else erro();
}
```

Solução: Fatoração à Esquerda !!!

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Problema 1

Considere GLC G: $S \rightarrow ab \mid ac$

FIRST$(ab) = \{a\}$, **FIRST**$(ac) = \{a\}$

FIRST$(ab) \cap$ **FIRST**$(ac) = \{a\} \neq \emptyset$

```
parse_S() {
    if (lookahead == 'a') {
        match('a'); match('b');          // S → ab
    } else if (lookahead == 'a') {
        match('a'); match('c');          // S → ac
    } else erro();
}
```

Solução: Fatoração à Esquerda !!!

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Problema 1

Considere GLC G: $S \rightarrow ab \mid ac$

FIRST$(ab) = \{a\}$, **FIRST**$(ac) = \{a\}$

FIRST$(ab) \cap$ **FIRST**$(ac) = \{a\} \neq \emptyset$

```
parse_S() {
    if (lookahead == 'a') {
        match('a'); match('b');          // S → ab
    } else if (lookahead == 'a') {
        match('a'); match('c');          // S → ac
    } else erro();
}
```

Solução: Fatoração à Esquerda !!!

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Problema 1

Considere GLC G: $S \rightarrow ab \mid ac$

FIRST$(ab) = \{a\}$, **FIRST**$(ac) = \{a\}$

FIRST$(ab) \cap$ **FIRST**$(ac) = \{a\} \neq \emptyset$

```
parse_S() {
    if (lookahead == 'a') {
        match('a'); match('b');          // S → ab
    } else if (lookahead == 'a') {
        match('a'); match('c');          // S → ac
    } else erro();
}
```

Solução: Fatoração à Esquerda !!!

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Problema 1

Considere GLC G: $S \rightarrow ab \mid ac$

FIRST$(ab) = \{a\}$, **FIRST**$(ac) = \{a\}$

FIRST$(ab) \cap$ **FIRST**$(ac) = \{a\} \neq \emptyset$

```
parse_S() {
    if (lookahead == 'a') {
        match('a'); match('b');          // S → ab
    } else if (lookahead == 'a') {
        match('a'); match('c');          // S → ac
    } else erro();
}
```

Solução: Fatoração à Esquerda !!!

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Problema 1

Considere GLC G: $S \rightarrow ab \mid ac$

FIRST$(ab) = \{a\}$, **FIRST**$(ac) = \{a\}$

FIRST$(ab) \cap$ **FIRST**$(ac) = \{a\} \neq \emptyset$

```
parse_S() {
    if (lookahead == 'a') {
        match('a'); match('b');          // S → ab
    } else if (lookahead == 'a') {
        match('a'); match('c');          // S → ac
    } else erro();
}
```

Solução: Fatoração à Esquerda !!!

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Problema 1 (Solução)

Fatorando G, tem-se G': $S \rightarrow a\ L$

$L \rightarrow b \mid c$

$\textbf{FIRST}(S) = \textbf{FIRST}(aL) = \{a\}$

$\textbf{FIRST}(L) = \textbf{FIRST}(b) \cup \textbf{FIRST}(c) = \{b, c\}$

```
parse_S() {
    if (lookahead == 'a') {
        match('a');
        parse_L(); // S -> a L
    } else error();
}
```

```
parse_L() {
    if (lookahead == 'b') {
        match('b'); // L -> b
    } else if (lookahead == 'c') {
        match('c'); // L -> c
    } else error();
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Problema 1 (Solução)

Fatorando G, tem-se G': $S \rightarrow a\ L$

$L \rightarrow b \mid c$

$\textbf{FIRST}(S) = \textbf{FIRST}(aL) = \{a\}$

$\textbf{FIRST}(L) = \textbf{FIRST}(b) \cup \textbf{FIRST}(c) = \{b, c\}$

```
parse_S() {
    if (lookahead == 'a') {
        match('a');
        parse_L();      // S → a L
    } else erro();
}
```

```
parse_L() {
    if (lookahead == 'b') {
        match('b');     // L → b
    } else if (lookahead == 'c') {
        match('c');     // L → c
    } else erro();
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Problema 1 (Solução)

Fatorando G, tem-se G': $S \rightarrow a\ L$

$L \rightarrow b \mid c$

$\mathbf{FIRST}(S) = \mathbf{FIRST}(aL) = \{a\}$

$\mathbf{FIRST}(L) = \mathbf{FIRST}(b) \cup \mathbf{FIRST}(c) = \{b, c\}$

```
parse_S() {
    if (lookahead == 'a') {
        match('a');
        parse_L();    // S → a L
    } else erro();
}
```

```
parse_L() {
    if (lookahead == 'b') {
        match('b');    // L → b
    } else if (lookahead == 'c') {
        match('c');    // L → c
    } else erro();
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Problema 1 (Solução)

Fatorando G, tem-se G': $S \rightarrow a\ L$

$L \rightarrow b \mid c$

$\mathbf{FIRST}(S) = \mathbf{FIRST}(aL) = \{a\}$

$\mathbf{FIRST}(L) = \mathbf{FIRST}(b) \cup \mathbf{FIRST}(c) = \{b, c\}$

```
parse_S() {
    if (lookahead == 'a') {
        match('a');
        parse_L();    // S → a L
    } else erro();
}
```

```
parse_L() {
    if (lookahead == 'b') {
        match('b');    // L → b
    } else if (lookahead == 'c') {
        match('c');    // L → c
    } else erro();
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Problema 1 (Solução)

Fatorando G, tem-se G': $S \rightarrow a\ L$

$L \rightarrow b \mid c$

$\mathbf{FIRST}(S) = \mathbf{FIRST}(aL) = \{a\}$

$\mathbf{FIRST}(L) = \mathbf{FIRST}(b) \cup \mathbf{FIRST}(c) = \{b, c\}$

```
parse_S() {
    if (lookahead == 'a') {
        match('a');
        parse_L();    // S → a L
    } else erro();
}
```

```
parse_L() {
    if (lookahead == 'b') {
        match('b');    // L → b
    } else if (lookahead == 'c') {
        match('c');    // L → c
    } else erro();
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Problema 1 (Solução)

Fatorando G, tem-se G': $S \rightarrow a\ L$

$L \rightarrow b \mid c$

$\mathbf{FIRST}(S) = \mathbf{FIRST}(aL) = \{a\}$

$\mathbf{FIRST}(L) = \mathbf{FIRST}(b) \cup \mathbf{FIRST}(c) = \{b, c\}$

```
parse_S() {
    if (lookahead == 'a') {
        match('a');
        parse_L();    // S → a L
    } else erro();
}
```

```
parse_L() {
    if (lookahead == 'b') {
        match('b');   // L → b
    } else if (lookahead == 'c') {
        match('c');   // L → c
    } else erro();
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Problema 2

Considere GLC G: $S \rightarrow S\ a \mid \lambda$

$\textbf{FIRST}(S) = \textbf{FIRST}(Sa) \cup \textbf{FIRST}(\lambda) = \{a, \lambda\}$

```
parse_S() {
  if (lookahead == 'a') {
    parse_S(); match('a');      // S → S a
  } else
    ;                           // S → λ
}
```

Problema: Existe um loop infinito em parse_S quando lookahead for 'a'!

Solução: Transformação da Recursão à Esquerda

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Problema 2

Considere GLC G: $S \rightarrow S\ a \mid \lambda$

$\textbf{FIRST}(S) = \textbf{FIRST}(Sa) \cup \textbf{FIRST}(\lambda) = \{a, \lambda\}$

```
parse_S() {
    if (lookahead == 'a') {
        parse_S(); match('a');          // S → S a
    } else ;
}
```

Problema: Existe um loop infitino em parse_S, quando *lookahead* for 'a' !

Solução: Eliminação da Recursão à Esquerda !!!

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Problema 2

Considere GLC G: $S \rightarrow S\ a \mid \lambda$

$\mathbf{FIRST}(S) = \mathbf{FIRST}(Sa) \cup \mathbf{FIRST}(\lambda) = \{a, \lambda\}$

```
parse_S() {
    if (lookahead == 'a') {
        parse_S(); match('a');          // S → S a
    } else ;
}
```

Problema: Existe um loop infitino em parse_S, quando *lookahead* for 'a' !

Solução: Eliminação da Recursão à Esquerda !!!

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Problema 2

Considere GLC G: $S \rightarrow S\ a \mid \lambda$

$\textbf{FIRST}(S) = \textbf{FIRST}(Sa) \cup \textbf{FIRST}(\lambda) = \{a, \lambda\}$

```
parse_S() {
    if (lookahead == 'a') {
        parse_S(); match('a');          // S → S a
    } else ;
}
```

Problema: Existe um loop infitino em parse_S, quando *lookahead* for 'a' !

Solução: Eliminação da Recursão à Esquerda !!!

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Problema 2

Considere GLC G: $S \rightarrow S\ a \mid \lambda$

$\textbf{FIRST}(S) = \textbf{FIRST}(Sa) \cup \textbf{FIRST}(\lambda) = \{a, \lambda\}$

```
parse_S() {
    if (lookahead == 'a') {
        parse_S(); match('a');          // S → S a
    } else ;
}
```

Problema: Existe um loop infitino em parse_S, quando *lookahead* for 'a' !

Solução: Eliminação da Recursão à Esquerda !!!

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Problema 2

Considere GLC G: $S \rightarrow S\ a \mid \lambda$

$\textbf{FIRST}(S) = \textbf{FIRST}(Sa) \cup \textbf{FIRST}(\lambda) = \{a, \lambda\}$

```
parse_S() {
    if (lookahead == 'a') {
        parse_S(); match('a');          // S → S a
    } else ;
}
```

Problema: Existe um loop infitino em parse_S, quando *lookahead* for 'a' !

Solução: Eliminação da Recursão à Esquerda !!!

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Problema 2 (Solução)

Eliminando recursão de G, tem-se G': $S \rightarrow L$

$L \rightarrow a\ L \mid \lambda$

$\mathbf{FIRST}(L) = \mathbf{FIRST}(aL) \cup \mathbf{FIRST}(\lambda) = \{a, \lambda\}$

$\mathbf{FIRST}(S) = (\mathbf{FIRST}(L) - \{\lambda\}) \cup \{\lambda\} = \{a, \lambda\}$

```
parse_S() {
    if (lookahead == 'a') {
        parse_L(); // S -> L
    } else {
        // S -> λ
    }
}
```

```
parse_L() {
    if (lookahead == 'a') {
        match('a');
        parse_L(); // L -> a L
    } else {
        // L -> λ
    }
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Problema 2 (Solução)

Eliminando recursão de G, tem-se G': $S \rightarrow L$

$L \rightarrow a\ L \mid \lambda$

$\mathbf{FIRST}(L) = \mathbf{FIRST}(aL) \cup \mathbf{FIRST}(\lambda) = \{a, \lambda\}$

$\mathbf{FIRST}(S) = (\mathbf{FIRST}(L) - \{\lambda\}) \cup \{\lambda\} = \{a, \lambda\}$

```
parse_S() {
    if (lookahead == 'a') {
        parse_L();    // S → L
    } else ;
}
```

```
parse_L() {
    if (lookahead == 'a') {
        match('a');
        parse_L();    // L → a L
    } else ;          // L → λ
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Problema 2 (Solução)

Eliminando recursão de G, tem-se G': $S \rightarrow L$

$L \rightarrow a\ L \mid \lambda$

$\textbf{FIRST}(L) = \textbf{FIRST}(aL) \cup \textbf{FIRST}(\lambda) = \{a, \lambda\}$

$\textbf{FIRST}(S) = (\textbf{FIRST}(L) - \{\lambda\}) \cup \{\lambda\} = \{a, \lambda\}$

```
parse_S() {
    if (lookahead == 'a') {
        parse_L();    // S → L
    } else ;
}
```

```
parse_L() {
    if (lookahead == 'a') {
        match('a');
        parse_L();    // L → a L
    } else ;          // L → λ
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Problema 2 (Solução)

Eliminando recursão de G, tem-se G': $S \rightarrow L$

$L \rightarrow a\ L \mid \lambda$

$\mathbf{FIRST}(L) = \mathbf{FIRST}(aL) \cup \mathbf{FIRST}(\lambda) = \{a, \lambda\}$

$\mathbf{FIRST}(S) = (\mathbf{FIRST}(L) - \{\lambda\}) \cup \{\lambda\} = \{a, \lambda\}$

```
parse_S() {
    if (lookahead == 'a') {
        parse_L();    // S → L
    } else ;
}
```

```
parse_L() {
    if (lookahead == 'a') {
        match('a');
        parse_L();    // L → a L
    } else ;          // L → λ
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Problema 2 (Solução)

Eliminando recursão de G, tem-se G': $S \rightarrow L$

$L \rightarrow a\ L \mid \lambda$

$\mathbf{FIRST}(L) = \mathbf{FIRST}(aL) \cup \mathbf{FIRST}(\lambda) = \{a, \lambda\}$

$\mathbf{FIRST}(S) = (\mathbf{FIRST}(L) - \{\lambda\}) \cup \{\lambda\} = \{a, \lambda\}$

```
parse_S() {
    if (lookahead == 'a') {
        parse_L();    // S → L
    } else ;
}
```

```
parse_L() {
    if (lookahead == 'a') {
        match('a');
        parse_L();    // L → a L
    } else ;          // L → λ
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Problema 2 (Solução)

Eliminando recursão de G, tem-se G': $S \rightarrow L$

$L \rightarrow a\ L \mid \lambda$

$\mathbf{FIRST}(L) = \mathbf{FIRST}(aL) \cup \mathbf{FIRST}(\lambda) = \{a, \lambda\}$

$\mathbf{FIRST}(S) = (\mathbf{FIRST}(L) - \{\lambda\}) \cup \{\lambda\} = \{a, \lambda\}$

```
parse_S() {
    if (lookahead == 'a') {
        parse_L();    // S → L
    } else ;
}
```

```
parse_L() {
    if (lookahead == 'a') {
        match('a');
        parse_L();    // L → a L
    } else ;          // L → λ
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Outro Exemplo

Seja GLC G: $E \rightarrow E + T \mid T$
$T \rightarrow T * F \mid F$
$F \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \mid (E)$

Eliminando as recursões à esquerda, tem-se:

G': $E \rightarrow T\ E'$
$E' \rightarrow +\ T\ E' \mid \lambda$
$T \rightarrow F\ T'$
$T' \rightarrow *\ F\ T' \mid \lambda$
$F \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \mid (E)$

$\mathbf{FIRST}(E) = \mathbf{FIRST}(T) = \mathbf{FIRST}(F) = \{\ '(', 0, 1, \ldots, 9\}$
$\mathbf{FIRST}(E') = \{\ '+', \lambda\}$
$\mathbf{FIRST}(T') = \{\ '*', \lambda\}$

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Outro Exemplo

Seja GLC G: $E \rightarrow E + T \mid T$
$T \rightarrow T * F \mid F$
$F \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \mid (E)$

Eliminando as recursões à esquerda, tem-se:

G': $E \rightarrow T\ E'$
$E' \rightarrow +\ T\ E' \mid \lambda$
$T \rightarrow F\ T'$
$T' \rightarrow *\ F\ T' \mid \lambda$
$F \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \mid (E)$

$\textbf{FIRST}(E) = \textbf{FIRST}(T) = \textbf{FIRST}(F) = \{\ '(', 0, 1, \ldots, 9\}$

$\textbf{FIRST}(E') = \{\ '+', \lambda\}$

$\textbf{FIRST}(T') = \{\ '*', \lambda\}$

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Outro Exemplo

Seja GLC G: $E \rightarrow E + T \mid T$
$T \rightarrow T * F \mid F$
$F \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \mid (E)$

Eliminando as recursões à esquerda, tem-se:

G': $E \rightarrow T\ E'$
$E' \rightarrow +\ T\ E' \mid \lambda$
$T \rightarrow F\ T'$
$T' \rightarrow *\ F\ T' \mid \lambda$
$F \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \mid (E)$

$\textbf{FIRST}(E) = \textbf{FIRST}(T) = \textbf{FIRST}(F) = \{\ '(', 0, 1, \ldots, 9\}$

$\textbf{FIRST}(E') = \{\ '+', \lambda\}$

$\textbf{FIRST}(T') = \{\ '*', \lambda\}$

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Outro Exemplo

Seja GLC G: $E \rightarrow E + T \mid T$
$T \rightarrow T * F \mid F$
$F \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \mid (E)$

Eliminando as recursões à esquerda, tem-se:

G': $E \rightarrow T\ E'$
$E' \rightarrow +\ T\ E' \mid \lambda$
$T \rightarrow F\ T'$
$T' \rightarrow *\ F\ T' \mid \lambda$
$F \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \mid (E)$

$\textbf{FIRST}(E) = \textbf{FIRST}(T) = \textbf{FIRST}(F) = \{\ '(', 0, 1, \ldots, 9\}$
$\textbf{FIRST}(E') = \{\ '+', \lambda\}$
$\textbf{FIRST}(T') = \{\ '*', \lambda\}$

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Outro Exemplo (cont.)

Para as regras de G': $E \rightarrow \ T\ E'$

$E' \rightarrow +\ T\ E' \ \mid \ \lambda$

Como: $\textbf{FIRST}(E) = \{\ '(', 0, 1, \ldots, 9\}$

$\textbf{FIRST}(E') = \{\ '+', \lambda\}$

```
parse_E() {
    if (lookahead == '(' ||
        (lookahead == '0') ||
            :
        (lookahead == '9') ||
            parse_T();
            parse_E'(); // E → T E'
    } else erro();
}
```

```
parse_E'() {
    if (lookahead == '+') {
        match('+');
        parse_T(); // E' → + T E'
        parse_E'();
    } else ;           // E' → λ
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Outro Exemplo (cont.)

Para as regras de G': $E \rightarrow \ T\ E'$

$E' \rightarrow +\ T\ E' \mid \lambda$

Como: $\textbf{FIRST}(E) = \{\ '(', 0, 1, \ldots, 9\}$

$\textbf{FIRST}(E') = \{\ '+', \lambda\}$

```
parse_E() {
    if (lookahead == '(' ||
      (lookahead == '0') ||
          ⋮
      (lookahead == '9') ||
          parse_T();
          parse_E'(); // E → T E'
    } else erro();
}
```

```
parse_E'() {
    if (lookahead == '+') {
        match('+');
        parse_T(); // E' → + T E'
        parse_E'();
    } else ;           // E' → λ
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Outro Exemplo (cont.)

Para as regras de G': $E \rightarrow \ T\ E'$

$E' \rightarrow +\ T\ E' \mid \lambda$

Como: $\mathbf{FIRST}(E) = \{\ '(', 0, 1, \ldots, 9\}$

$\mathbf{FIRST}(E') = \{\ '+', \lambda\}$

```
parse_E() {
    if (lookahead == '(' ||
      (lookahead == '0') ||
          ⋮
      (lookahead == '9') ||
          parse_T();
          parse_E'(); // E → T E'
    } else erro();
}
```

```
parse_E'() {
    if (lookahead == '+') {
        match('+');
        parse_T(); // E' → + T E'
        parse_E'();
    } else ;            // E' → λ
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Outro Exemplo (cont.)

Para as regras de G': $T \rightarrow F\ T'$

$T' \rightarrow *\ F\ T' \mid \lambda$

Como: $\textbf{FIRST}(T) = \{\ '(', 0, 1, \ldots, 9\}$

$\textbf{FIRST}(T') = \{\ '*', \lambda\}$

```
parse_T() {
    if (lookahead == '(' ||
        lookahead == '0' ||
          :
        lookahead == '9' ||
          parse_F();
          parse_T'(); // T → F T'
    } else erro();
}
```

```
parse_T'() {
    if (lookahead == '*') {
        match('*');
        parse_F();  // T' → * F T'
        parse_T'();
    } else ;        // T' → λ
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Outro Exemplo (cont.)

Para as regras de G': $T \rightarrow F\ T'$

$T' \rightarrow *\ F\ T' \mid \lambda$

Como: $\mathbf{FIRST}(T) = \{\ '(', 0, 1, \ldots, 9\}$

$\mathbf{FIRST}(T') = \{\ '*', \lambda\}$

```
parse_T() {
    if (lookahead == '(' ||
        (lookahead == '0') ||
            ⋮
        (lookahead == '9') ||
            parse_F();
            parse_T'(); // T → F T'
    } else erro();
}
```

```
parse_T'() {
    if (lookahead == '*') {
        match('*');
        parse_F(); // T' → * F T'
        parse_T'();
    } else ; // T' → λ
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Outro Exemplo (cont.)

Para as regras de G': $T \rightarrow F\ T'$

$T' \rightarrow *\ F\ T' \mid \lambda$

Como: $\textbf{FIRST}(T) = \{\ '(', 0, 1, \ldots, 9\}$

$\textbf{FIRST}(T') = \{\ '*', \lambda\}$

```
parse_T() {
    if (lookahead == '(' ||
      (lookahead == '0') ||
          :
      (lookahead == '9') ||
          parse_F();
          parse_T'(); // T → F T'
    } else erro();
}
```

```
parse_T'() {
    if (lookahead == '*') {
        match('*');
        parse_F();  // T' → * F T'
        parse_T'();
    } else ;            // T' → λ
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Outro Exemplo (cont.)

Para as regras de G': $F \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \mid (E)$
Como: $\textbf{FIRST}(F) = \{\ '(', 0, 1, \ldots, 9\}$

```
parse_F() {
    if (lookahead == '(') {
        match('(');
        parse_E();  // F → (E)
        match(')');
    } else if (lookahead == '0') {
        match('0');  // F → 0
    } else if (lookahead == '1') {
        :
    } else if (lookahead == '9') {
        match('9');  // F → 9
    } else erro();
}
```

Parsing Preditivo Recursivo

Parser Recursivo Descendente – Outro Exemplo (cont.)

Para as regras de G': $F \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \mid (E)$
Como: **FIRST**$(F) = \{\ '(', 0, 1, \ldots, 9\}$

```
parse_F() {
    if (lookahead == '(') {
        match('(');
        parse_E();  // F → (E)
        match(')');
    } else if (lookahead == '0') {
        match('0');  // F → 0
    } else if (lookahead == '1') {
        :
    } else if (lookahead == '9') {
        match('9');  // F → 9
    }  else erro();
}
```