Atividade Prática: Construção e Implementação de um Parser Preditivo Tabular LL(1)

Objetivo

- Compreender-se a construção dos conjuntos FIRST e FOLLOW para uma gramática dada.
- Elaborar-se a tabela de análise LL(1) com base nesses conjuntos.
- Analisar-se uma entrada manualmente para entender o processo de parsing.
- Implementar-se um parser LL(1) para verificar a análise.

Gramática da Atividade

Considera-se a seguinte gramática simples para expressões condicionais:

```
S → if E then S else S | a
E → b | c
```

Onde:

- `S` é o símbolo inicial (sentença).
- Terminais: `if`, `then`, `else`, `a`, `b`, `c`, `$` (fim da entrada).
- Não-terminais: `S`, `E`.

Instruções Detalhadas

Etapa 1: Construção dos Conjuntos FIRST

Os conjuntos FIRST indicam os terminais que podem iniciar uma derivação a partir de cada não-terminal. Seguem-se as regras para sua construção:

1. Para uma produção $A \to X_1\ X_2\ \ldots\ X_n$:
    - Incluem-se os terminais de $FIRST(X_1)$ em $FIRST(A)$.
    - Se X_1 deriva ϵ, adicionam-se os de $FIRST(X_2)$, e assim por diante.
    - Se todos derivam ϵ, inclui-se ϵ em $FIRST(A)$.

Passos:

1. **FIRST(S)**:
    - Produções: `S → if E then S else S` e `S → a`.
    - Para `S → if E then S else S`, o primeiro símbolo é `if` → Inclui-se `{if}`.

- Para `S → a`, o primeiro símbolo é `a` → Inclui-se `{a}`.
- Resultado: `FIRST(S) = {if, a}`.

2. **FIRST(E)**:
   - Produções: `E → b` e `E → c`.
   - Para `E → b`, o primeiro símbolo é `b` → Inclui-se `{b}`.
   - Para `E → c`, o primeiro símbolo é `c` → Inclui-se `{c}`.
   - Resultado: `FIRST(E) = {b, c}`.

Resultado Esperado:

- `FIRST(S) = {if, a}`
- `FIRST(E) = {b, c}`

Tarefa: Anota-se os conjuntos FIRST calculados em um documento ou caderno.

Etapa 2: Construção dos Conjuntos FOLLOW

Os conjuntos FOLLOW indicam os terminais que podem aparecer após cada não-terminal em uma derivação válida. Seguem-se as regras:

1. Para o símbolo inicial `S`, adiciona-se `$`.
2. Para `A → α B β`, adicionam-se os terminais de `FIRST(β)` (exceto `ϵ`) a `FOLLOW(B)`.
3. Se `β` deriva `ϵ` ou `A → α B`, adiciona-se `FOLLOW(A)` a `FOLLOW(B)`.

Passos:

1. **FOLLOW(S)**:
   - `S` é o símbolo inicial → Inclui-se `$`.
   - Produção `S → if E then S else S`:
     - Após o primeiro `S`, vem `else S`. Inclui-se `FIRST(else S) = {else}` em `FOLLOW(S)`.
     - Após o segundo `S`, nada segue diretamente, mas `FOLLOW(S)` já inclui `$` por ser o símbolo inicial.
   - Resultado: `FOLLOW(S) = {else, $}`.
2. **FOLLOW(E)**:
   - Produção `S → if E then S else S`: Após `E`, vem `then S else S`. Inclui-se `FIRST(then S else S) = {then}` em `FOLLOW(E)`.
   - Resultado: `FOLLOW(E) = {then}`.

Resultado Esperado:

- `FOLLOW(S) = {else, $}`
- `FOLLOW(E) = {then}`

Tarefa: Anota-se os conjuntos FOLLOW calculados.

Etapa 3: Construção da Tabela LL(1)

A tabela é preenchida com base nos conjuntos FIRST e FOLLOW:

- Para cada produção `A → α`:
  - Se `t ∈ FIRST(α)` (e `t ≠ ϵ`), insere-se `A → α` em `[A, t]`.
  - Se `ϵ ∈ FIRST(α)`, insere-se `A → α` em `[A, t]` para todo `t ∈ FOLLOW(A)`.

Passos:

1. **Tabela para S**:
   - Produção `S → if E then S else S`: `FIRST = {if}` → Insere-se em `[S, if]`.
   - Produção `S → a`: `FIRST = {a}` → Insere-se em `[S, a]`.
2. **Tabela para E**:
   - Produção `E → b`: `FIRST = {b}` → Insere-se em `[E, b]`.
   - Produção `E → c`: `FIRST = {c}` → Insere-se em `[E, c]`.

Tabela Esperada:

	if	a	b	c	then	else	$
S	S→ifEthenSelseS	S→a					
E			E→b	E→c			

Tarefa: Constrói-se a tabela em papel ou editor de texto com base nos cálculos.

Etapa 4: Processo de Parsing Manual

Analisa-se a entrada `if b then a else a $` manualmente utilizando a tabela e uma pilha. O processo segue estas etapas:

1. Inicia-se com a pilha `[$ S]` e o lookahead como o primeiro token da entrada.
2. Para cada iteração:
    - Se o topo da pilha for um não-terminal, consulta-se a tabela `[topo, lookahead]` e aplica-se a produção, empilhando os símbolos em ordem inversa.
    - Se o topo for um terminal, verifica-se se corresponde ao lookahead; em caso positivo, remove-se o topo e avança-se o lookahead.
    - Repete-se até que a pilha esteja vazia e o lookahead seja `$`.

Exemplo de Tabela de Análise:

Pilha	Entrada	Ação
$ S	if b then a else a $	[S, if] → S→ifEthenSelseS
$ S else S then E if	if b then a else a $	Match if
$ S else S then E	b then a else a $	[E, b] → E→b
$ S else S then b	b then a else a $	Match b
$ S else S then	then a else a $	Match then
$ S else S	a else a $	[S, a] → S→a
$ S else a	a else a $	Match a
$ S else	else a $	Match else
$ S	a $	[S, a] → S→a
$ a	a $	Match a
$	$	Match $

Resultado Esperado: A entrada é aceita, pois a pilha fica vazia e o lookahead é `$`.

Tarefa: Registra-se cada passo da análise em uma tabela semelhante.

Etapa 5: Implementação do Parser

Implementa-se um parser LL(1) para a gramática fornecida, com base na tabela construída. O parser deve:

- Utilizar uma pilha para rastrear os símbolos.
- Consultar a tabela LL(1) para aplicar produções.
- Comparar terminais com o lookahead e avançar na entrada.
- Exibir os passos do processo (pilha, entrada e ação) para fins didáticos.

Orientações para a Implementação:

1. **Estrutura Básica**:
    - Define-se a tabela como uma estrutura de dados que mapeia pares `(não-terminal, terminal)` para listas de símbolos das produções.
    - Cria-se uma função ou mecanismo para distinguir não-terminais de terminais.
    - Implementa-se a lógica do parser em um programa principal.
2. **Passos do Algoritmo**:
    - Inicia-se a pilha com `[$ S]` (símbolo inicial e fim).
    - Adiciona-se `$` ao final da entrada.
    - Enquanto a pilha não estiver vazia:
        - Se o topo for um terminal ou `$`, verifica-se correspondência com o lookahead; se houver, remove-se o topo e avança-se.
        - Se o topo for um não-terminal, consulta-se a tabela e aplica-se a produção, empilhando os símbolos em ordem inversa.
        - Registra-se cada ação (ex.: exibição ou log).
    - Verifica-se se a entrada foi aceita (pilha vazia e lookahead `$`).
3. **Representação da Tabela**:
    - Baseada na tabela construída:
        - `[S, if] → [if, E, then, S, else, S]`
        - `[S, a] → [a]`
        - `[E, b] → [b]`
        - `[E, c] → [c]`

Tarefa:

1. Escolhe-se uma linguagem de programação de preferência (ex.: Python, C, Java).
2. Define-se a tabela em uma estrutura adequada (ex.: dicionário, matriz, array associativo).
3. Implementa-se a lógica do parser seguindo o algoritmo descrito.
4. Testa-se o programa com a entrada `if b then a else a`.

5. Garante-se que o programa exiba os passos (pilha, entrada e ação) e informe se a entrada foi aceita ou rejeitada.

Dica: Pode-se usar uma estrutura de pilha nativa da linguagem escolhida e uma representação tabular que facilite consultas rápidas.
