senão, sai da subrotina L_ID sem chamar a rotina de ERRO, significando o reconhecimento de L_ID → ε.

3.2.3 Análise Preditiva Tabular

É possível construir analisadores preditivos *não recursivos* que utilizam uma pilha explícita ao invés de chamadas recursivas (pilha implícita). Esse tipo de analisador implementa um autômato de pilha controlado por uma tabela de análise. O princípio do reconhecimento preditivo é a determinação da produção a ser aplicada, cujo lado direito irá substituir o símbolo não–terminal que se encontra no topo da pilha. O analisador busca a produção a ser aplicada na tabela de análise, levando em conta o não-terminal no topo da pilha e o *token* sob o cabeçote de leitura.

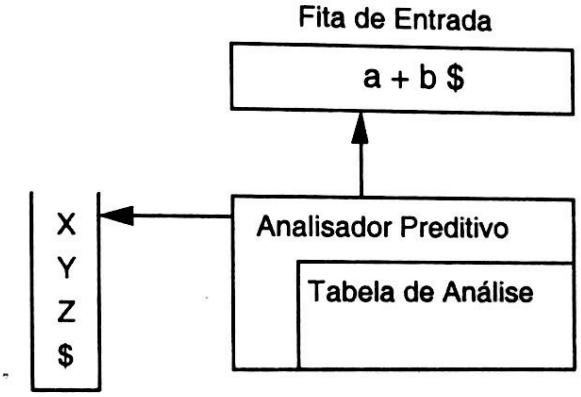

Figura 3.4 Analisador preditivo tabular

Um analisador preditivo orientado por tabela compreende uma fita de entrada, uma pilha e uma tabela de análise, conforme é mostrado na Figura 3.4. A fita de entrada contém a sentença a ser analisada seguida de **$**, símbolo que marca o fim da sentença. Inicialmente, a pilha contém **$**, que marca a sua base, seguido do símbolo inicial da gramática. A tabela de análise é uma matriz **M** com **n** linhas e **t+1** colunas, onde **n** é o número de símbolos não–terminais, e **t** é o número de símbolos terminais (a coluna extra corresponde ao símbolo **$**).

O analisador é controlado por um programa que se comporta conforme descrito a seguir. Considerando **X** como o símbolo no topo da pilha e **a** como terminal da fita de entrada sob o cabeçote de leitura, o analisador executa uma de três ações possíveis:

1) se **X = a = $**, o analisador pára, aceitando a sentença;

2) se **X = a ≠ $**, o analisador desempilha **a** e avança o cabeçote de leitura para o próximo símbolo na fita de entrada;

3) se X é um símbolo não–terminal, o analisador consulta a entrada M[X, a] da tabela de análise. Essa entrada poderá conter uma produção da gramática ou ser vazia. Supondo M[X, a] = {X → UVW}, o analisador substitui X (que está no topo da pilha) por WVU (ficando U no topo) e retorna a produção aplicada. Se M[X, a] é vazia, isso corresponde a uma situação de erro; nesse caso, o analisador chama uma rotina de tratamento de erro.

O comportamento do analisador pode ser descrito através de uma tabela que mostra, a cada passo, o conteúdo da pilha e o restante da sentença a ser lida, conforme é exemplificado a seguir.

EXEMPLO 3.10 Movimentos de um analisador tabular preditivo.

Considere a gramática não-ambígua abaixo que gera expressões lógicas:

$$E \rightarrow E \vee T \mid T$$
$$T \rightarrow T \& F \mid F$$
$$F \rightarrow \neg F \mid id$$

Eliminando-se a recursividade à esquerda das produções que definem E e T, obtém-se:

$$E \rightarrow T\,E'$$
$$E' \rightarrow \vee T\,E' \mid \varepsilon$$
$$T \rightarrow F\,T'$$
$$T' \rightarrow \& F\,T' \mid \varepsilon$$
$$F \rightarrow \neg F \mid id$$

A tabela de análise preditiva para essa gramática é mostrada a seguir:

	id	∨	&	¬	$
E	E → T E'			E → T E'	
E'		E' → ∨T E'			E' → ε
T	T → F T'			T → F T'	
T'		T' → ε	T' → & F T'		T' → ε
F	F → id			F → ¬F	

Com a sentença **idvid&id**, o reconhecedor preditivo realiza os movimentos mostrados na Figura 3.5.

Inicialmente, o cabeçote aponta para o símbolo mais à esquerda da sentença de entrada. Observando as ações produzidas pelo reconhecedor, pode-se notar que as produções usadas na análise constituem uma derivação mais à esquerda da sentença.

Pilha	Entrada	Ação
$E	id ∨ id & id $	E → T E'
$E' T	id ∨ id & id $	T → F T'
$E' T' F	id ∨ id & id $	F → id
$E' T' id	id ∨ id & id $	desempilha e lê símbolo
$E' T'	∨ id & id $	T' → ε
$E'	∨ id & id $	E' → ∨ T E'
$E' T ∨	∨ id & id $	desempilha e lê símbolo
$E' T	id & id $	T → F T'
$E' T' F	id & id $	F → id
$E' T' id	id & id $	desempilha e lê símbolo
$E' T'	& id $	T' → & F T'
$E' T' F &	& id $	desempilha e lê símbolo
$E' T' F	id $	F → id
$E' T' id	id $	desempilha e lê símbolo
$E' T'	$	T' → ε
$E'	$	E' → ε
$	$	Aceita a sentença!

Figura 3.5 Movimentos de um analisador tabular preditivo

O algoritmo que guia os movimentos de um analisador preditivo não–recursivo é apresentado a seguir:

Algoritmo do Analisador Preditivo Tabular:

Entrada: Uma sentença **s** e a tabela de análise **M** para a gramática G.

Resultado: Uma derivação mais à esquerda de **s,** se **s** está em L(G), ou uma indicação de erro, caso contrário.

Método: Inicialmente, o analisador está numa configuração na qual a pilha contém **$S** (com S no topo), e a fita de entrada contém **s$**. O programa utiliza a tabela de análise preditiva **M** e comporta-se do seguinte modo:

Posiciona o cabeçote sobre o primeiro símbolo de **s$;**

Seja **X** o símbolo do topo da pilha e **a** o símbolo sob o cabeçote.

```
Repete
    se X é um terminal
    então se X = a
            então desempilha X e avança o cabeçote
            senão ERRO
    senão /* X é um símbolo não–terminal */
            se M[X,a] = X → Y1 Y2 ... Yk
            então { desempilha X;
                    empilha YkYk-1...Y1 com Y1 no topo;
                    imprime a produção X → Y1Y2...Yk }
            senão ERRO
até que X = $  /* pilha vazia */
```

❑

Na implementação de um analisador preditivo tabular, a maior dificuldade está na construção da tabela de análise. Para construir essa tabela, é necessário computar duas funções associadas à gramática: as funções **FIRST** e **FOLLOW**.

Definição 3.10 FIRST(α).
Se α é uma forma sentencial (seqüência de símbolos da gramática), então **FIRST(α)** é o conjunto de terminais que iniciam formas sentenciais derivadas a partir de α. Se $\alpha \Rightarrow^* \varepsilon$, então a palavra vazia também faz parte do conjunto.

Definição 3.11 FOLLOW(A).
A função **FOLLOW** é definida para símbolos não–terminais. Sendo **A** um não–terminal, ***FOLLOW(A)*** é o conjunto de terminais **a** que podem aparecer imediatamente à direita de **A** em alguma forma sentencial. Isto é, o conjunto de terminais **a**, tal que existe uma derivação da forma $S \Rightarrow^* \alpha A a \beta$ para α e β quaisquer.

Cálculo das funções FIRST e FOLLOW

Algoritmo para calcular FIRST(X):

Para computar **FIRST(X)** para um símbolo **X** da gramática, aplicam-se as regras abaixo, até que não se possa adicionar mais terminais ou ε ao conjunto em questão.

1) Se **a** é terminal, então **FIRST(a) = { a }**.

2) Se $X \rightarrow \varepsilon$ é uma produção, então adicione ε a **FIRST(X)**.

3) Se $X \rightarrow Y_1 Y_2 ... Y_k$ é uma produção e, para algum i, todos $Y_1, Y_2, ..., Y_{i-1}$ derivam ε, então **FIRST(Y_i)** está em **FIRST(X)**. Se todo Y_j (j =1,2,...,k) deriva ε, então ε está em **FIRST(X)**.

Algoritmo para calcular FOLLOW(X):

Para computar **FOLLOW(X)**, aplicam-se as regras abaixo até que não se possa adicionar mais símbolos ao conjunto.

1) Se **S** é o símbolo inicial da gramática e **$** é o marcador de fim da sentença, então **$** está em **FOLLOW(S)**.

2) Se existe produção do tipo $A \rightarrow \alpha X \beta$, então todos os terminais de **FIRST(β)** fazem parte de **FOLLOW(X)**.

3) Se existe produção do tipo $A \rightarrow \alpha X$, ou $A \rightarrow \alpha X \beta$, sendo que $\beta \Rightarrow^* \varepsilon$, então todos os terminais que estiverem em **FOLLOW(A)** fazem parte de **FOLLOW(X)**.

EXEMPLO 3.11 Determinação das funções FIRST e FOLLOW.

Considere novamente a gramática da expressão lógica:

$$E \rightarrow T\,E'$$
$$E' \rightarrow \vee\, T\,E' \mid \varepsilon$$
$$T \rightarrow F\,T'$$
$$T' \rightarrow \&\, F\,T' \mid \varepsilon$$
$$F \rightarrow \neg F \mid id$$

Conjuntos FIRST:

Convém iniciar pelos não–terminais para os quais a obtenção do conjunto FIRST é trivial. Isso ocorre para F, que deriva apenas formas sentenciais que iniciam por terminais:

$$FIRST(F) = \{ \neg, id \}$$

T' deriva no terminal & e na palavra vazia; logo:

$$FIRST(T') = \{ \&, \varepsilon \}$$

Similarmente, E' deriva no terminal ∨ e na palavra vazia:

$$FIRST(E') = \{ \vee, \varepsilon \}$$

Como T deriva apenas em FT', e F não deriva a palavra vazia, então FIRST(T) = FIRST(F). Portanto:

$$FIRST(T) = \{ \neg, id \}$$

Como E deriva apenas em E → TE' e T não deriva ε, tem-se FIRST(E) = FIRST(T):

$$FIRST(E) = \{ \neg , id \}$$

Conjuntos FOLLOW:

FOLLOW(E) contém $ pela regra 1:

$$FOLLOW(E) = \{ \$ \}$$

Tem-se que FOLLOW(E') = FOLLOW(E), pois E' é o ultimo símbolo do lado direito da produção E → TE' (regra 3):

$$FOLLOW(E') = \{ \$ \}$$

A partir da análise da produção E' → ∨TE', FOLLOW(T) é obtido pela união de FIRST(E'), pela regra 2, com FOLLOW(E'), pela regra 3, pois E'→ ε.

$$FOLLOW(T) = \{ \vee , \$ \}$$

FOLLOW(T') = FOLLOW(T), pois T' é último na produção T → FT':

$$FOLLOW(T') = \{ \vee, \$ \}$$

FOLLOW(F) é obtido pela união de FIRST(T'), pela regra 2, com FOLLOW(T'), pela regra 3:

$$FOLLOW(F) = \{ \vee , \& , \$ \}$$

❑

O algoritmo para a construção da tabela de análise é apresentado a seguir.

Algoritmo para construir uma tabela de análise preditiva:

Entrada: gramática G

Resultado: Tabela de Análise M

Método:

1) Para cada produção A → α de G, execute os passos 2 e 3 (para criar a linha A da tabela M).
2) Para cada terminal **a** de **FIRST(α)**, adicione a produção **A → α** a **M[A,a]**.
3) Se **FIRST(α)** inclui a palavra vazia, então adicione **A → α** a **M[A, b]** para cada **b** em **FOLLOW(A)**.

E::=TE'
E'::=∨TE' | ε
T::=FT'
T'::=&FT' | ε
F::=¬F | id

Aplicando o algoritmo acima à gramática que gera expressões lógicas, obtém-se as seguintes entradas para a tabela de análise. (A tabela é reapresentada abaixo.)

Para	$E \rightarrow T\ E'$	tem-se	FIRST(T E') = { ¬, id }	M[E, ¬] = M[E, id] = E→ T E'
Para	$E' \rightarrow \vee\ T\ E'$	tem-se	FIRST(∨T E') = { ∨ }	M[E', ∨] = E → ∨ T E'
Para	$E' \rightarrow \varepsilon$	tem-se	FOLLOW(E') = { $ }	M[E',$] = E'→ ε
Para	$T \rightarrow FT'$	tem-se	FIRST(F T') = { ¬, id }	M[T, ¬] = M[T, id] = T→ F T'
Para	$T' \rightarrow \&\ F\ T'$	tem-se	FIRST(& FT) = { & }	M[T',&] = T' → & F T'
Para	$T' \rightarrow \varepsilon$	tem-se	FOLLOW(T') = { ∨, $}	M[T', ∨] = M[T',$]= T'→ ε
Para	$F \rightarrow \neg F$	tem-se	FIRST((¬ F)) = { ¬ }	M[F, ¬] = F → ¬ F
Para	$F \rightarrow id$	tem-se	FIRST(id) = { id }	M[F, id] = F → id

FIRST:
E{¬,id}
E'{∨, ε}
T{¬, id}
T'{&, ε}
F{¬, id}

FOLLOW:
E{$}
E'{$}
T{∨,$}
T'{∨,$}
F{∨,&,$}

	id	∨	&	¬	$
E	E → T E'			E → T E'	
E'		E' → ∨ T E'			E' → ε
T	T → F T'			T → F T'	
T'		T' → ε	T' → & F T'		T' → ε
F	F → id			F → ¬ F	

Se, em cada entrada da Tabela de Análise, existe apenas uma produção, então a gramática que originou a tabela é dita ser do tipo LL(1). Isso significa que as sentenças geradas pela gramática são passíveis de serem analisadas da esquerda para a direita (Left to right), produzindo uma derivação mais à esquerda (Leftmost derivation), levando em conta apenas um (1) símbolo da entrada. Essas gramáticas são mais bem caracterizadas adiante, na Definição 3.12.

O algoritmo anterior pode ser aplicado a qualquer gramática para produzir a tabela de análise. Contudo, para certas gramáticas, a tabela resultante poderá ter entradas multiplamente definidas. Isso ocorre para as gramáticas ambíguas, consequentemente, essas gramáticas não são do tipo LL(1).

EXEMPLO 3.12 Tabela sintática para uma gramática ambígua.

Seja G a gramática abaixo:

$$S \rightarrow \text{if } C \text{ then } S\ S' \mid a$$
$$S' \rightarrow \text{else } S \mid \varepsilon$$
$$C \rightarrow b$$