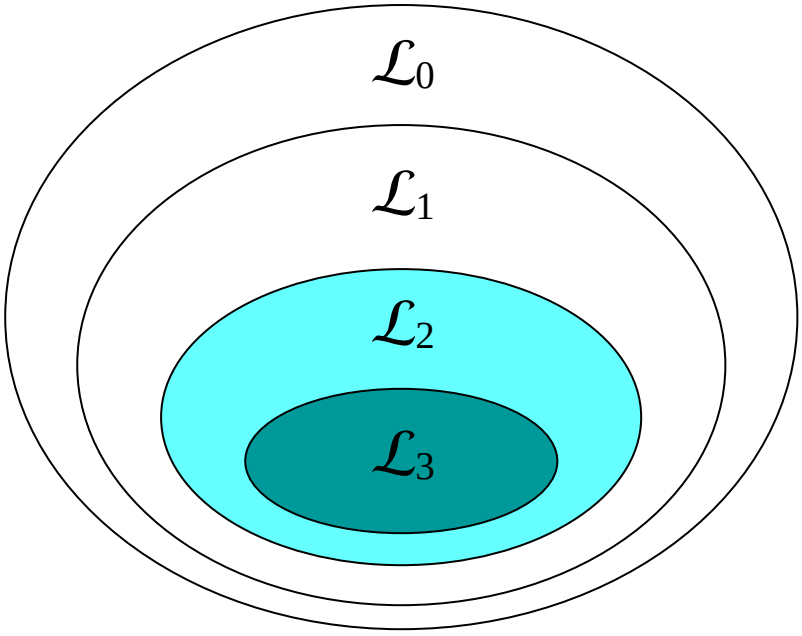
Capítulo IV :
Linguagens Independentes do Contexto

Uma Classificação de Linguagens e respectivos Autómatos:

A Hierarquia de Chomsky

	Linguagem	Autómato Reconhecedor
$\mathcal{L}_0$	Linguagem com Estrutura de Frase	Autómato de duas Pilhas (Máquina de Turing)
$\mathcal{L}_1$	Linguagem Sensível ao Contexto	Autómato de duas Pilhas (Máquina de Turing)
$\mathcal{L}_2$	Linguagem Independente do Contexto	Autómato de uma Pilha
$\mathcal{L}_3$	**Linguagem Regular**	**Autómato Finito**



Gramáticas Geradoras de Linguagens:

Definição: Uma **Gramática Geradora** é um quádruplo ordenado,

$$G = (V, T, P, S)$$

onde:

- V e T são conjunto finitos não vazios e disjuntos, onde V é o conjunto dos símbolos não terminais ou **variáveis** e T o conjunto dos **símbolos terminais**.
  $V \cup T$ chama-se **vocabulário** completo da Gramática.
- P é um conjunto finito de **produções** ou regras de rescrita.
- $S \in V$ é o símbolo **inicial** ou Axioma da Gramática.

Definição: Uma Gramática Geradora diz-se **Independente do Contexto** quando cada produção tem a forma,

$$A \rightarrow \alpha \quad \text{onde} \quad A \in V \ \text{ e } \ \alpha \in (V \cup T)^*$$

Exemplo: A gramática $G = (\{S\}, \{0, 1\}, P, S)$ com as produções:

$$S \rightarrow \varepsilon$$
$$S \rightarrow 0$$
$$S \rightarrow 1$$
$$S \rightarrow 0S0$$
$$S \rightarrow 1S1$$

define os palíndromos sobre $\{0, 1\}$. Geralmente abreviamos,

$$S \rightarrow \varepsilon \mid 0 \mid 1 \mid 0S0 \mid 1S1$$

Exemplo: Uma Gramática de **Expressões Aritméticas** (E), sobre Números (N) Binários e Identificadores (I), só com as letras a e b, usando apenas os Operadores + e $*$,

$G = (\{E, I, N\}, \{a, b, 0, 1, (,), +, *, -\}, P, E)$ com

$$\begin{aligned} E &\rightarrow I \mid N \mid E + E \mid E * E \mid (E) \\ I &\rightarrow a \mid b \mid Ia \mid Ib \\ N &\rightarrow 0 \mid 1 \mid N0 \mid N1 \mid - N \mid + N \end{aligned}$$

Será que a palavra $(a + ab) * -100$ é uma Expressão Aritmética?

$$\begin{aligned} E &\Rightarrow E * E \Rightarrow E * N \Rightarrow E * - N \Rightarrow E * - N0 \Rightarrow E * - N00 \\ &\Rightarrow E * -100 \Rightarrow (E) * -100 \Rightarrow (E + E) * -100 \Rightarrow (E + I) * -100 \\ &\Rightarrow (E + Ib) * -100 \Rightarrow (E + ab) * -100 \Rightarrow (I + ab) * -100 \\ &\Rightarrow (a + ab) * -100 \end{aligned}$$

Partindo do símbolo inicial e por **derivações de um só passo**, é possível atingir uma sequência de símbolos terminais, igual à palavra dada.

Notações:

- **Derivação num só passo:**

  Numa dada Gramática, sejam $A \in V$ e $\alpha, \beta, \gamma \in (V \cup T)^*$

  dizemos que $\alpha A \beta \Rightarrow \alpha \gamma \beta$ quando existe uma produção $A \rightarrow \gamma$

- **Derivação** (em zero ou mais passos)**:**

  $\alpha \Rightarrow^* \beta$ … ou melhor, $\alpha \overset{*}{\Rightarrow} \beta$

Definição: **Linguagem Gerada** por uma Gramática G = (V, T, P, S):

$$L(G) = \{w \in T^* \mid S \Rightarrow^* w\}$$

Definição: Uma **Linguagem** L diz-se **Independente do Contexto** se existir alguma Gramática G, Independente do Contexto, tal que L = L(G).

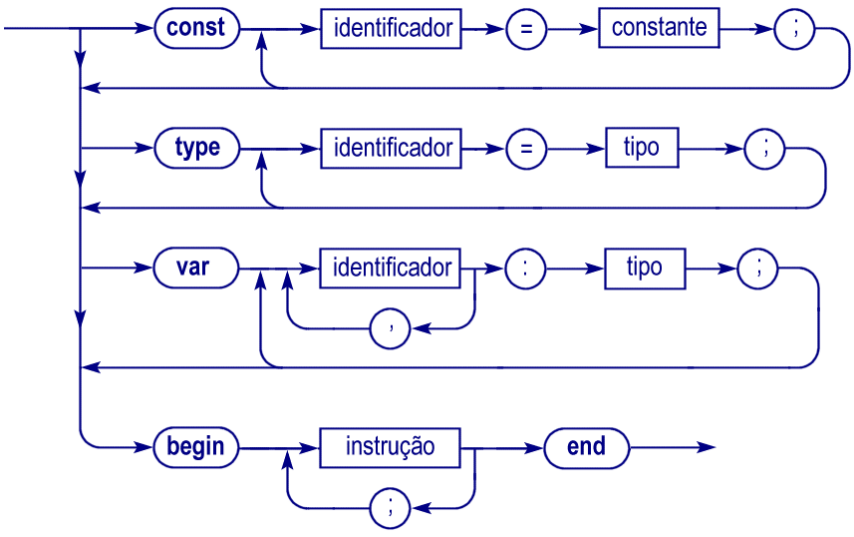
Aplicação: Uma LIG para as **instruções compostas** em C++

```
<compound stmt> --> { <stmt list> }
<stmt list> --> <stmt> <stmt list> | epsilon
<stmt> --> <compound stmt>
<stmt> --> if ( <expr> ) <stmt>
<stmt> --> if ( <expr> ) <stmt> else <stmt>
<stmt> --> while ( <expr> ) <stmt>
<stmt> --> do <stmt> while ( <expr> ) ;
<stmt> --> for ( <stmt> <expr> ; <expr> ) <stmt>
<stmt> --> case <expr> : <stmt>
<stmt> --> switch ( <expr> ) <stmt>
<stmt> --> break ; | continue ;
<stmt> --> return <expr> ; | goto <id> ;
```

Aplicação: Uma LIG para **Nmtoken** em XML

```
NameChar    ::=   Letter | Digit | '.' | '-' | '_' | ':' |
                  CombiningChar | Extender
Name        ::=   (Letter | '_' | ':') (NameChar)*
Names       ::=   Name (S Name)*
Nmtoken     ::=   (NameChar)+
Nmtokens    ::=   Nmtoken (S Nmtoken)*
```

Aplicação: Uma LIG para o **bloco** de um programa em Pascal



Como verificar que uma Gramática gera a Linguagem pretendida?

Consideremos a gramática $G = (\{S\}, \{0, 1\}, P, S)$ com
$S \longrightarrow \varepsilon \mid 0 \mid 1 \mid 0S0 \mid 1S1$

e seja $Pal = \{w \in \{0, 1\}^* \mid w = w^R\}$.

Provemos que **Pal = L(G)**.

Pal ⊆ L(G): Para qualquer $w = w^R$ provemos (por Indução sobre $|w|$) que $w \in L(G)$, isto é, que $S \Rightarrow^* w$.

Caso Base: Se $|w| = 0$ ou $|w| = 1$, então $w = \varepsilon$ ou 0 ou 1 e $S \longrightarrow \varepsilon \mid 0 \mid 1$.

Passo Indutivo: (H.I.: Se $x = x^R$ com $|x| \leq n$, então $S \Rightarrow^* x$)
Se $w = w^R$ com $|w| = n+1 \geq 2$ então w é da forma $w = 0x0$ ou $w = 1x1$ com $x = x^R$.
Por Hipótese de Indução $S \Rightarrow^* x$, portanto
$S \Rightarrow 0S0 \Rightarrow^* 0x0$ ou $S \Rightarrow 1S1 \Rightarrow^* 1x1$,
ou seja $S \Rightarrow^* w$.

L(G) ⊆ Pal: Seja $w \in L(G)$, isto é, $S \Rightarrow^* w$. Provemos, por Indução sobre o número de passos de derivação que $w = w^R$.

Caso Base: As derivações num só passo são e $S \longrightarrow \varepsilon$, $S \longrightarrow 0$ e $S \longrightarrow 1$. Então $w = \varepsilon$ ou 0 ou $1 \in Pal$.

Passo Indutivo: (H.I.: Se $S \Rightarrow^* x$ em $\underline{n}$ passos então $x = x^R$)
Uma derivação com $\underline{n+1}$ passos de w deverá ser da forma $S \Rightarrow 0S0 \Rightarrow^* 0x0 = w$ ou da forma $S \Rightarrow 1S1 \Rightarrow^* 1x1 = w$.
Em ambos os casos como, por Hipótese de Indução $x = x^R$, então $w = w^R$.

Retomemos a Gramática de Expressões Aritméticas e consideremos duas sequências possíveis de derivações para (a + b) ∗ 10,

Seguindo sempre a **derivação** (mais) **à esquerda**:

$$
\begin{aligned}
E &\Rightarrow E * E \\
&\Rightarrow (E) * E \\
&\Rightarrow (E + E) * E \\
&\Rightarrow (I + E) * E \\
&\Rightarrow (a + E) * E \\
&\Rightarrow (a + I) * E \\
&\Rightarrow (a + b) * E \\
&\Rightarrow (a + b) * N \\
&\Rightarrow (a + b) * N0 \\
&\Rightarrow (a + b) * 10
\end{aligned}
$$

$E \Rightarrow^*_{lm} (a + b) * 10$ (leftmost)

Seguindo sempre a **derivação** (mais) **à direita**:

$$
\begin{aligned}
E &\Rightarrow E * E \\
&\Rightarrow E * N \\
&\Rightarrow E * N0 \\
&\Rightarrow E * 10 \\
&\Rightarrow (E) * 10 \\
&\Rightarrow (E + E) * 10 \\
&\Rightarrow (E + I) * 10 \\
&\Rightarrow (E + b) * 10 \\
&\Rightarrow (a + b) * 10 \\
&\Rightarrow (a + b) * 10
\end{aligned}
$$

$E \Rightarrow^*_{rm} (a + b) * 10$ (rightmost)

Proposição: $A \Rightarrow^* w$ sse $A \Rightarrow^*_{lm} w$ e $A \Rightarrow^*_{rm} w$.

Uma sequência de símbolos terminais e não terminais de uma Gramática, (designado habitualmente por uma letra grega) $\alpha \in (V \cup T)^*$ diz-se uma:

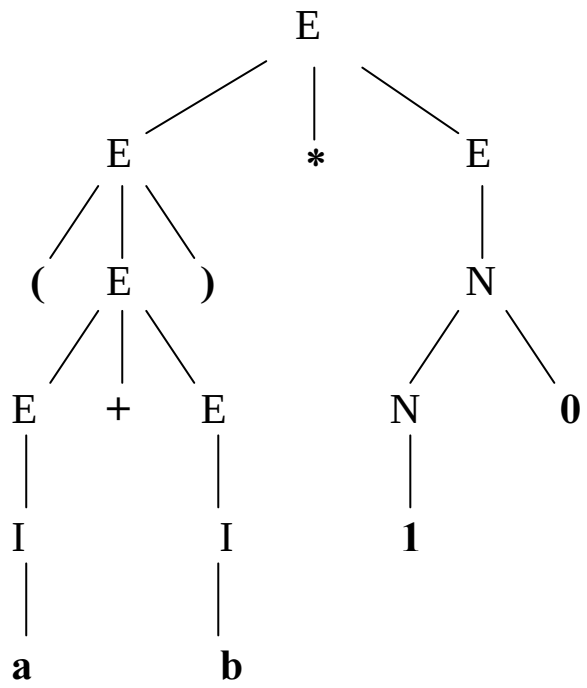
- forma de frase, se $S \Rightarrow^* \alpha$
- forma de frase à esquerda, se $S \Rightarrow^*_{lm} \alpha$
- forma de frase à direita, se $S \Rightarrow^*_{rm} \alpha$

Árvores de Derivação

Para uma Gramática G = (V, T, P, S), a derivação de uma palavra é representada graficamente por uma estrutura de árvore, onde:

- a raiz representa S;
- cada vértice interno representa uma variável de V;
- cada folha representa ε ou um símbolo terminal de T;
- se um vértice interno, representando uma variável A tem como filhos os vértices $X_1, X_2, ..., X_n$ então existe em P uma produção $A \rightarrow X_1 X_2 ... X_n$.

Exemplo:



- O **valor** da árvore de derivação de (a + b) ∗ 10 consiste na concatenação dos símbolos terminais nas folhas da árvore de derivação, caminhando da esquerda para a direita.
- A estrutura da árvore não especifica qual foi a ordem por que foram efectuadas as derivações.
- Neste caso, a árvore é a mesma, quer se tenham seguido derivações à esquerda ou à direita.

Equivalência entre Árvores de Derivação e Derivações

Dados uma Gramática Independente do Contexto G = (V, T, P, S), um símbolo não terminal $A \in V$ e uma palavra $w \in T^*$, prova-se que são equivalentes:

1. $A \Rightarrow^* w$ (se A = S, então $w \in L(G)$)
2. $A \Rightarrow^*_{lm} w$
3. $A \Rightarrow^*_{rm} w$
4. Existe alguma Árvore de Derivação com raiz A e valor *w*.

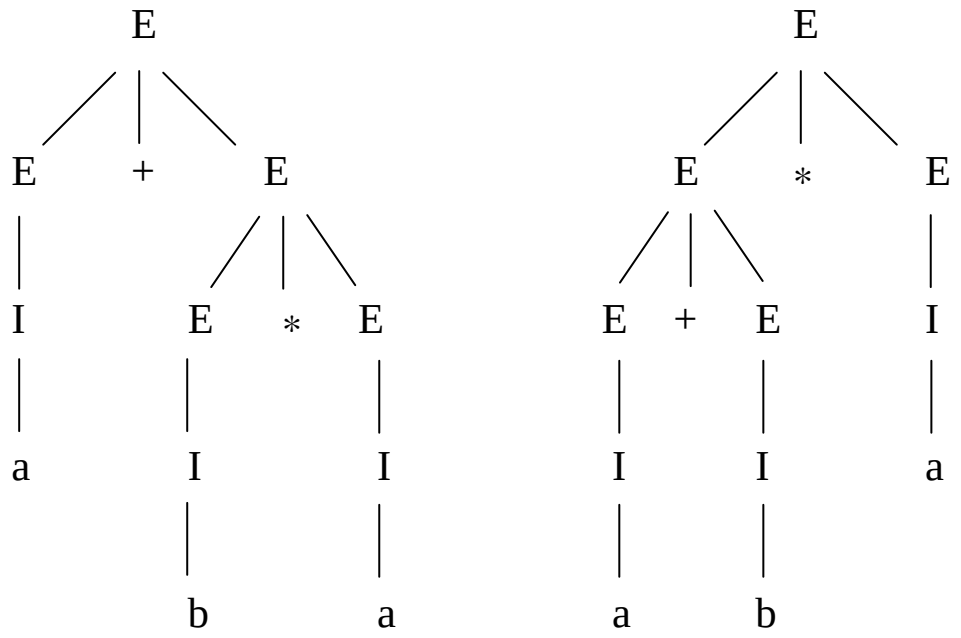
Ambiguidades em Gramáticas e em Linguagens

Gramáticas ambíguas:

Para a Gramática de Expressões Aritméticas, consideremos as seguintes derivações para a + b ∗ a,

$$
\begin{aligned}
E &\Rightarrow E + E \\
&\Rightarrow I + E \\
&\Rightarrow a + E \\
&\Rightarrow a + E * E \\
&\Rightarrow a + I * E \\
&\Rightarrow a + b * E \\
&\Rightarrow a + b * I \\
&\Rightarrow a + b * a
\end{aligned}
\qquad
\begin{aligned}
E &\Rightarrow E * E \\
&\Rightarrow E + E * E \\
&\Rightarrow I + E * E \\
&\Rightarrow a + E * E \\
&\Rightarrow a + I * E \\
&\Rightarrow a + b * E \\
&\Rightarrow a + b * I \\
&\Rightarrow a + b * a
\end{aligned}
$$

Mesmo seguindo sempre derivações à esquerda, são geradas duas árvores de derivação distintas.
Tratando-se do cálculo do valor de uma expressão aritmética as duas derivações conduzem, efectivamente, a dois resultados diferentes.



Definição: Uma **Gramática** G = (V, T, P, S) diz-se **ambígua** quando existe, pelo menos, uma cadeia $w \in T^*$ para a qual existem duas árvores de derivação distintas.

Remoção da ambiguidade:

A ambiguidade das expressões aritméticas resulta do facto de a Gramática não estipular as regras habituais de precedência dos operadores.

Neste caso, é fácil construir uma **Gramática Equivalente** (que gera a mesma Linguagem) que não é ambígua. A precedência do produto em relação à soma é imposta pela introdução das variáveis Termo e Factor.

$$E \rightarrow T \mid E + T$$

$$T \rightarrow F \mid T * F$$

$$F \rightarrow I \mid N \mid (E)$$

$$N \rightarrow 0 \mid 1 \mid N0 \mid N1 \mid - N \mid + N$$

$$I \rightarrow a \mid b \mid Ia \mid Ib$$

Outro, bem conhecido, exemplo consiste na ambiguidade provocada por,

```
<stmt> --> if <expr> then <stmt>
         | if <expr> then <stmt> else <stmt>
         | other
```

que é habitualmente resolvida por,

```
<stmt> --> <match_stmt>
         | <unmatch_stmt>
<match_stmt> --> if <expr> then <match_stmt> else <match_stmt>
               | other
<unmatch_stmt> --> if <expr> then <stmt>
                 | if <expr> then <match_stmt> else <unmatch_stmt>
                 | other
```

Verifique que, com esta Gramática, a instrução Pascal,

if E1 **then if** E2 **then** S1 **else** S2

admite uma única árvore de derivação.

Definição: Uma **Linguagem** diz-se **Inerentemente Ambígua** quando <u>todas</u> as suas Gramáticas Geradoras são ambíguas.

Exemplo: Prova-se que a Linguagem Independente do Contexto,

$$L = \{a^n b^n c^m d^m \mid n, m \geq 1\} \cup \{ a^n b^m c^m d^n \mid n, m \geq 1\}$$

é inerentemente ambígua. Para qualquer Gramática Geradora de L, uma cadeia do tipo $a^k b^k c^k d^k$ tem sempre duas árvores de derivação possíveis, correspondentes à identificação do conjunto a que pertence.

Notas:

- A maioria das linguagens de programação incluem detalhes que não são, efectivamente, Independentes do Contexto. Por exemplo,

  $L = \{a^n b^m c^n d^m \mid n, m \geq 1\}$

  que não é uma Linguagem Independente do Contexto, destina-se a formalizar a concordância entre o número de parâmetros na definição e na chamada de um sub-programa.

- As Linguagens Regulares nunca podem ser inerentemente ambíguas. Cada LR é definida por algum AFD onde o reconhecimento de cada palavra corresponde a um caminho único.

- Prova-se que toda a Linguagem Regular pode ser gerada por uma Gramática G = (V, T, P, S) com apenas dois tipos de produções:

  1. $X \rightarrow a\,Y$ com $X, Y \in V$ e $a \in T$
  2. $X \rightarrow \varepsilon$ com $X \in V$

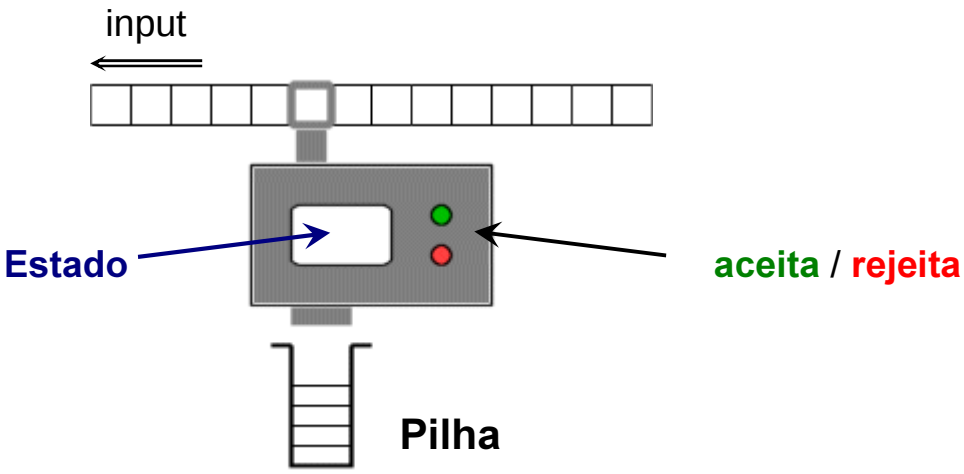
Problemas de Decisão sobre Linguagens Independentes do Contexto:

? Encontrar uma Gramática não ambígua equivalente a uma dada Gramática.
Não existe Algoritmo para resolver este Problema.

? Verificar se uma dada Gramática é, ou não, ambígua.
Nem para este.

Autómato de (uma) Pilha

Tomemos um Autómato Finito (AFND-ε) e juntemos uma **Pilha**, de grandeza ilimitada, onde o Autómato pode colocar e retirar itens.



⇨ A ideia:

Reconhecimento de: $\{ a^n b^n \mid \forall\ n \geq 0 \}$

- Para cada a lido, colocá-lo na Pilha;
- Para cada b lido, remover um a da Pilha;

{ a palavra é aceite se e só se a Pilha estiver vazia no fim do processo }

Reconhecimento de: $\{ w\ c\ w^R \mid w \in (a + b)^* \}$

- Ir lendo a's e b's, colocando-os na Pilha;
- Até aparecer o c;
- Ir lendo a's e b's e removendo-os na Pilha, verificando se são iguais;

{ a palavra é aceite se e só se a Pilha estiver vazia no fim do processo }

Reconhecimento de: $\{ w\ w^R \mid w \in (a + b)^* \}$

- Ir lendo a's e b's, colocando-os na Pilha;
- Decisão não-determinista (!) ;
- Ir lendo a's e b's e removendo-os na Pilha, verificando se são iguais;

{ a palavra é aceite se e só se a Pilha estiver vazia no fim do processo }

Como funciona?

Ler um símbolo de entrada e o item no topo da Pilha e, com base nesses valores:

1. Remover o item do topo da Pilha;
2. Colocar um novo item na Pilha;
3. Transitar para outro estado;
4. Avançar a janela de leitura.

Exemplo: Consideremos a Linguagem Independente do Contexto,

$L = \{ww^R \mid w \in (0 + 1)^*\}$

que representa os palíndromos binários de comprimento par.

Esta linguagem é gerada por $G = (\{S\}, \{0, 1\}, P, S)$ com,

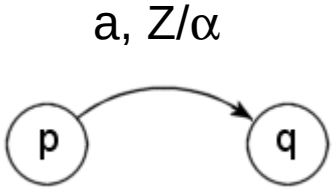
$$S \rightarrow \varepsilon$$
$$S \rightarrow 0S0$$
$$S \rightarrow 1S1$$

Reconhecimento por um Autómato (ND-ε) de Pilha:

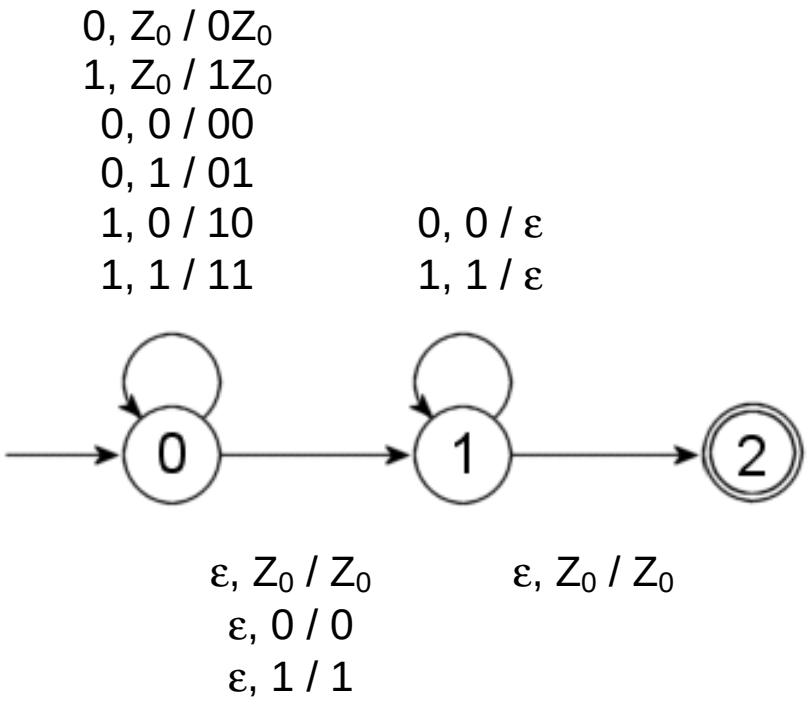
1. Começando no estado inicial q_0, ir lendo cada símbolo de entrada e colocando-o na Pilha;
2. A certa altura (transição ND-ε) transitar para o estado q_1;
3. Continuar a ler os símbolos de entrada e, ao mesmo tempo, ir removendo itens da Pilha verificando se são iguais;
4. Se a Pilha ficar vazia ao mesmo tempo que é lido o último símbolo, então a palavra é reconhecida.

Representação gráfica de uma transição:

"A partir do estado p,
com o símbolo de entrada a
e o item Z no topo da Pilha,
transitar para o estado q,
substituindo Z por α no topo da Pilha
e avançar para o próximo símbolo de entrada."



Para o exemplo anterior:



Simulação para w = 1001:

símbolos que falta ler	**Pilha**	**Estado**	
1001	Z_0	q_0	
001	$1Z_0$	q_0	
01	$01Z_0$	q_0	
01	$01Z_0$	q_1	
1	$1Z_0$	q_1	
ε	Z_0	q_1	
ε	Z_0	q_2	**aceitação**

Definição: Um **Autómato de uma Pilha** é um séptuplo ordenado,

$$P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

onde:

- Q é um conjunto finito, não vazio, de **estados internos**,
- Σ é um alfabeto finito de **símbolos de entrada**,
- Γ é um alfabeto finito de **símbolos da Pilha**,
- δ é uma aplicação de $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma$ no conjunto dos subconjuntos finitos de $Q \times \Gamma^*$ chamada função de **transição** ou de **mudança de estado**,
- $Z_0 \in \Gamma \setminus \Sigma$ é o **símbolo inicial** da Pilha.
- $q_0 \in Q$ é o **estado inicial**,
- $F \subseteq Q$ é o conjunto dos **estados finais** ou de aceitação.

A Função de Transição:

- Para $a \in \Sigma$, a transição $\delta(q, a, Z) = \{(p_1, \gamma_1), (p_2, \gamma_2), ..., (p_k, \gamma_k)\}$ significa: a partir do estado q, por leitura do símbolo a, com o item Z no topo da Pilha, escolher (não deterministicamente) um i,
  1. transitar para p_i;
  2. remover Z;
  3. colocar γ_i na Pilha;
  4. avançar para o próximo símbolo de entrada.

- Para $a = \varepsilon$, a transição $\delta(q, \varepsilon, Z) = \{(p_1, \gamma_1), (p_2, \gamma_2), ..., (p_k, \gamma_k)\}$ significa: a partir do estado q, com o item Z no topo da Pilha, qualquer que seja o símbolo de entrada (e mesmo que já estejam todos lidos), escolher (não-deterministicamente) um i,
  1. transitar para p_i;
  2. remover Z;
  3. colocar γ_i na Pilha;
  4. **não** avançar na cadeia de entrada.

Exemplo: Consideremos a Gramática Independente do Contexto,

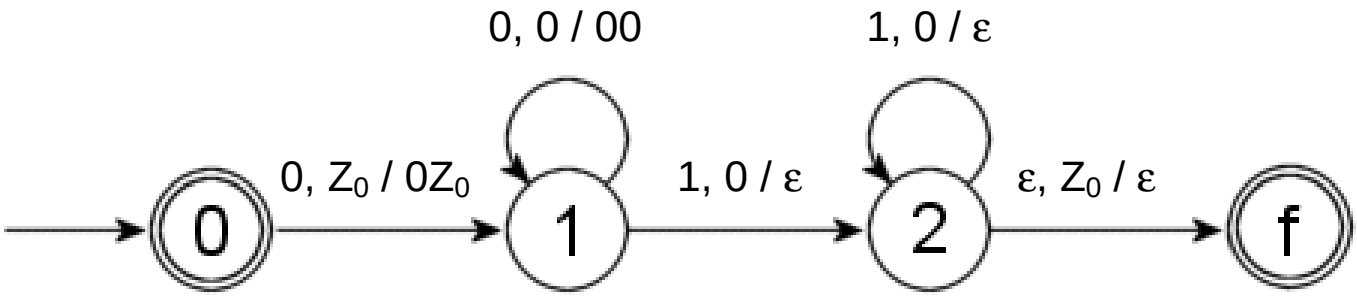
$$G = (\{S\}, \{0, 1\}, \{ S \rightarrow 0S1, S \rightarrow \varepsilon\}, S)$$

geradora da Linguagem,

$$L = \{0^n 1^n \mid n \geq 0\}.$$

Consideremos também o Autómato de Pilha,

$$P = (\{q_0, q_1, q_2, q_f\}, \{0, 1\}, \{0, 1, Z_0\}, \delta, q_0, Z_0, \{q_0, q_f\})$$



e analisemos as cinco transições possíveis:

	símbolo	operação na Pilha	transição
$\delta(q_0, 0, Z_0) = \{(q_1, 0Z_0)\}$	0	colocar 0	para q_1
$\delta(q_1, 0, 0) = \{(q_1, 00)\}$	0	colocar 0	
$\delta(q_1, 1, 0) = \{(q_2, \varepsilon)\}$	1	remover 0	para q_2
$\delta(q_2, 1, 0) = \{(q_2, \varepsilon)\}$	1	remover 0	
$\delta(q_2, \varepsilon, Z_0) = \{(q_f, \varepsilon)\}$		remover Z_0	para q_f

(No caso de um Autómato Finito bastava, em cada instante, conhecer o seu Estado)

Definição: A descrição instantânea de um Autómato de Pilha é dada pela sua **configuração** (q, w, γ) onde,

1. q é o presente Estado;
2. w a sequência de símbolos que falta ler;
3. γ o conteúdo da Pilha.

(No caso de um Autómatos Finito, a extensão $\delta^{\wedge}$ bastava para representar sequências de transições)

Definição: Um **passo** de um Autómato de Pilha é representado por,

$(q, aw, Z\alpha) \vdash (p, w, \beta\alpha)$, sempre que $(p, \beta) \in \delta(q, a, Z)$,

e $\vdash^*$ representa uma sequência de passos.

(O reconhecimento (aceitação) de uma palavra pode agora acontecer de **dois** modos)

Linguagens Reconhecidas por Autómatos de Pilha:

Para um Autómato de Pilha $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, definimos:

1. Aceitação por Estado Final:

$$\mathbf{L(P)} = \{w \in \Sigma^* \mid \exists\, p \in F, \exists\, \alpha \in \Gamma^* : (q_0, w, Z_0) \vdash^* (p, \varepsilon, \alpha)\}$$

Partindo do estado inicial, P lê toda a palavra e atinge um estado final.

2. Aceitação por Pilha Vazia:

$$\mathbf{N(P)} = \{w \in \Sigma^* \mid \exists\, p \in Q : (q_0, w, Z_0) \vdash^* (p, \varepsilon, \varepsilon)\}$$

Partindo do estado inicial, P lê toda a palavra e esvazia a Pilha.

Equivalência das noções de Aceitação:

1. Se uma linguagem L = L(P_1) para algum Autómato de Pilha P_1, então existe um Autómato de Pilha P_2 para o qual L = N(P_2).

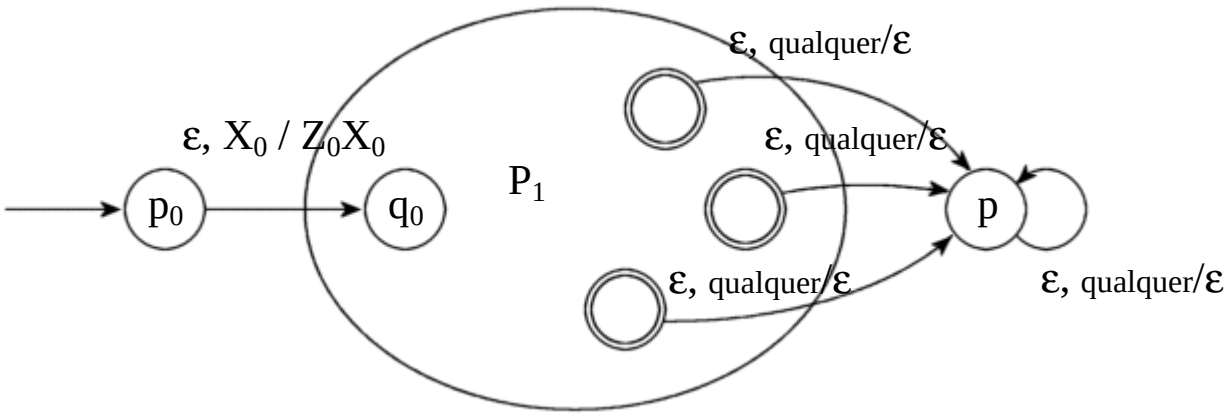
2. Se uma linguagem L = N(P_1) para algum Autómato de Pilha P_1, então existe um Autómato de Pilha P_2 para o qual L = L(P_2).

Demonstração / Construção:

1. Seja $P_1 = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, para o qual L = L($P_1$).
   Vamos construir P_2, para o qual L = N(P_2):

   > P_2 deve simular o funcionamento de P_1 de modo que, quando P_1 atinja um estado final, P_2 esvazie a Pilha. O problema consiste em evitar que P_1 esvazie a sua Pilha, antes de atingir um estado final.

   Introduzir e um novo símbolo inicial X_0 para a Pilha de P_2 e um novo estado inicial p_0, cuja função é colocar o símbolo inicial de P_1 e passar ao estado inicial de P_1.



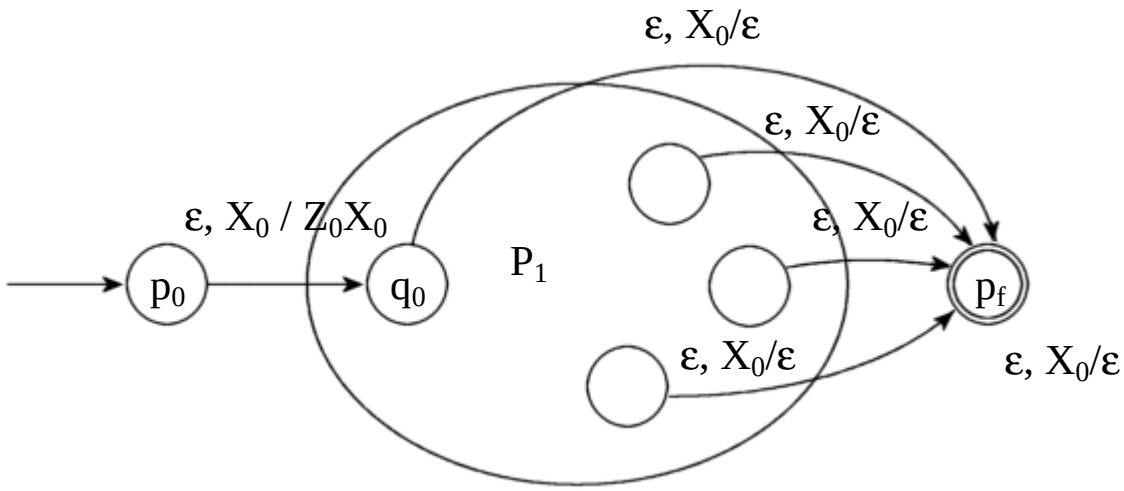
   P_2 simula todas as transições de P_1, até aos estados finais de P_1.

   Introduzir um novo estado p, onde a Pilha de P_2 é esvaziada.
   Se um estado final de P_1 for atingido, a Pilha de P_2 é esvaziada sem que seja lido mais nenhum símbolo da cadeia de entrada.

2. Seja $P_1 = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, para o qual $L = N(P_1)$.
Vamos construir P_2, para o qual $L = L(P_2)$:

> P_2 deve simular o funcionamento de P_1 de modo que, quando P_1 esvazie a Pilha, P_2 atinja um estado final.

Introduzir e um novo símbolo inicial X_0 e um novo estado inicial p_0.



P_2 simula todas as transições de P_1, até aos estados em que P_1 esvazia a Pilha. Introduzir p_f como único estado final de P_2.
Sempre que a Pilha de P_1 é detectada vazia, ocorre uma transição para o estado de aceitação de P_2.

Equivalência entre Gramáticas Independentes do Contexto e Autómatos de uma Pilha:

1. Linguagens geradas por Gramáticas Independentes do Contexto.
2. Linguagens reconhecidas por Estado Final de algum Autómato de Pilha.
3. Linguagens reconhecidas por Pilha Vazia de algum Autómato de Pilha.

Comecemos por verificar que:

L = L(G) → L = N(P)

Dada uma Linguagem **L = L(G)**, gerada por alguma Gramática G = (V, Σ, R, S) Independente do Contexto, construir um Autómato de Pilha tal que **L = N(P)**, isto é, P reconhece L por Pilha Vazia.

Estratégia:

Construir P de modo a simular todas as derivações possíveis, no reconhecimento de uma palavra de L.

Irão ocorrer dois tipos de passos no Autómato:

- Se um símbolo terminal a ∈ Σ for lido e estiver no topo da Pilha, então é removido da Pilha;
- Se uma variável A ∈ V estiver no topo da Pilha, então será substituída pelo lado direito de cada produção que tenha A no lado esquerdo.

Deste modo, basta que o Autómato pretendido tenha um só estado. Tratando-se de uma aceitação por Pilha Vazia, não é necessário considerar Estados Finais.

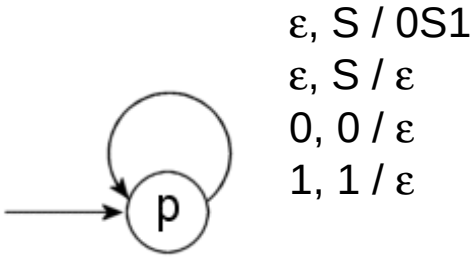
Exemplo: Para a GIC geradora de $\{0^n1^n \mid n \geq 0\}$:

$G = (\{S\}, \{0, 1\}, \{S \rightarrow 0S1 \mid \varepsilon\}, S)$

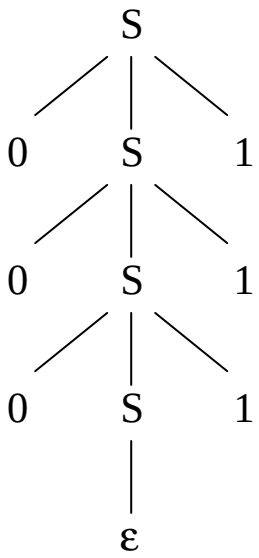
O Autómato P tal que N(P) = L(G):

P = ({p}, {0, 1}, {0, 1, S}, δ, p, S, {}), com δ definido por:

$\delta(p, \varepsilon, S) = \{(p, 0S1), (p, \varepsilon)\}$
$\delta(p, 0, 0) = \{(p, \varepsilon)\}$
$\delta(p, 1, 1) = \{(p, \varepsilon)\}$



Simulação para *w* = 000111



000111	S
000111	0S1
00111	S1
00111	0S11
0111	S11
0111	0S111
111	S111
111	111
11	11
1	1
ε	

Construção do Autómato:

P = ({q}, Σ, V ∪ Σ, δ, q, S, {}) tal que:

P tem apenas um Estado, não tem Estados Finais, o Alfabeto da Pilha é o Alfabeto completo de G, onde S funciona como Símbolo Inicial. A Função de Transição é definida por:

- se a ∈ Σ, então δ(q, a, a) = {(q, ε)};
- se A ∈ V, então δ(q, ε, A) = {(q, α) | ∃ A → α em P}.

Exemplo:

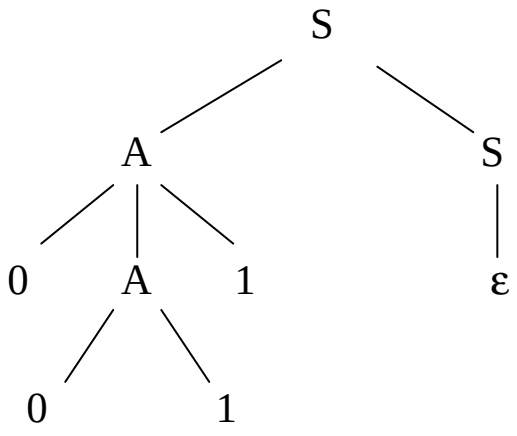
G = ({S, A}, {0, 1}, R, S), com R:

$$S \rightarrow AS \mid \varepsilon$$
$$A \rightarrow 0A1 \mid A1 \mid 01$$

P = ({q}, {0, 1}, {0, 1, A, S}, δ, q, S, {}), com δ definido por:

$$\delta(q, \varepsilon, S) = \{(q, AS), (q, \varepsilon)\}$$
$$\delta(q, \varepsilon, A) = \{(q, 0A1), (q, A1), (q, 01)\}$$
$$\delta(q, 0, 0) = \{(q, \varepsilon)\}$$
$$\delta(q, 1, 1) = \{(q, \varepsilon)\}$$

Reconhecimento de *w* = 0011:



símbolos que falta ler	transição	Pilha	Derivação
0011		S	S ⇒
0011	(q, AS)	AS	⇒ AS
0011	(q, 0A1)	0A1S	⇒ 0A1S
0011	(q, ε)	A1S	
011	(q, 01)	011S	⇒ 0011S
011	(q, ε)	11S	
11	(q, ε)	1S	
1	(q, ε)	S	⇒ 0011
ε	(q, ε)	ε	

Exemplo: Uma Gramática de Expressões Aritméticas (E) sobre Identificadores (I) construídos só com as letras a e b e os dígitos binários, usando apenas os Operadores + e ∗,

G = ({E, I}, {a, b, 0, 1, (,), +, ∗}, P, E) com

$$E \rightarrow I \mid E + E \mid E * E \mid (E)$$
$$I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$$

O alfabeto da Pilha do Autómato pretendido é

{a, b, 0, 1, (,), +, ∗, E, I}

e a função de transição definida por:

$\delta(q, \varepsilon, E) = \{(q, I), (q, E + E), (q, E * E), (q, (E))\}$

$\delta(q, \varepsilon, I) = \{(q, a), (q, b), (q, Ia), (q, Ib), (q, I0), (q, I1)\}$

$\delta(q, a, a) = \{(q, \varepsilon)\}$
$\delta(q, b, b) = \{(q, \varepsilon)\}$
$\delta(q, 0, 0) = \{(q, \varepsilon)\}$
$\delta(q, 1, 1) = \{(q, \varepsilon)\}$
$\delta(q, (, () = \{(q, \varepsilon)\}$
$\delta(q,),)) = \{(q, \varepsilon)\}$
$\delta(q, +, +) = \{(q, \varepsilon)\}$
$\delta(q, *, *) = \{(q, \varepsilon)\}$

Verifiquemos agora que:

L = N(P) → L = L(G)

Dada uma Linguagem L, para a qual existe algum Autómato de Pilha $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \{\})$, tal que L = N(P), construir uma Gramática Independente do Contexto G = (V, Σ, R, S), tal que L = L(G).

Estratégia:

Cada passo do Autómato envolve as acções de: por leitura de algum símbolo, remover $A \in \Gamma$ da Pilha, colocar algum $\gamma \in \Gamma$ na Pilha e transitar de um estado $p \in Q$ para um estado $q \in Q$.

O Autómato reconhece a palavra $w \in \Sigma^*$,
se e só se, esvazia a Pilha por leitura de w.

Como representar as variáveis de G?

O terno **[pAq]** representa a acção de: partindo do estado p, com o símbolo A no topo da Pilha, transitar para o estado q, removendo A.

Os ternos [pAq] serão as **variáveis** da gramática G pretendida, aos quais juntamos o símbolo de partida S.

A variável [pAq] gera as palavras $w \in \Sigma^*$,
tais que $(p, w, A) \vdash^* (q, \varepsilon, \varepsilon)$,
ou seja $[pAq] \Rightarrow^* w$

Como representar as produções de G?

Partindo dos ternos $[q_0 Z_0 p]$, $\forall\ p \in Q$, começamos por construir as produções:

$$S \rightarrow [q_0 Z_0 p] \quad \text{(uma para cada estado } p \in Q\text{)},$$

além disso, sempre que $(p_1, Y_1 Y_2 \ldots Y_k) \in \delta(p, a, X)$,
onde a pode ser ε e k pode ser 0, então
para todos os possíveis estados $p_2, p_3, \ldots, p_{k+1} \in Q$,
juntar também as produções:

$$[p X p_{k+1}] \rightarrow a\ [p_1 Y_1 p_2]\ [p_2 Y_2 p_3] \ldots [p_k Y_k p_{k+1}].$$

Por exemplo,

Se $(q, \varepsilon) \in \delta(p, a, X)$, ou seja, remover X da Pilha por leitura de a, juntar a produção:

$$[pXq] \rightarrow a$$

Se $(q, Y) \in \delta(p, a, X)$, ou seja, mudar de estado e substituir o elemento X por Y no topo da Pilha. Para todos os estados r, juntar as produções:

$$[pXr] \rightarrow a\ [qYr]$$

Se $(q, YZ) \in \delta(p, a, X)$, ou seja, mudar de estado e substituir o elemento X no topo da Pilha por YZ. Para todos os estados r e s, juntar:

$$[pXr] \rightarrow a\ [qYs]\ [sZr]$$

Exemplo: Consideremos a Sintaxe da Instrução `if` na linguagem C.

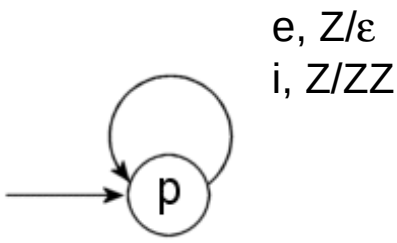
```
<stmt> --> if ( <expr> ) <stmt>
<stmt> --> if ( <expr> ) <stmt> else <stmt>
```

Analisemos o seguinte problema associado:

Verificar se ocorrem mais `else`'s do que `if` 's.

Comecemos por construir um Autómato de Pilha, capaz de verificar essa ocorrência, **por Pilha Vazia**.

Colocamos mais um símbolo Z na Pilha, sempre que aparece um `if` e removemos um Z sempre que ocorre um `else`.
O esvaziamento da Pilha indica que apareceu um `else` a mais.



O Autómato P_N pretendido é definido por:

$$P_N = (\{p\}, \{i, e\}, \{Z\}, \delta_N, p, Z, \{\})$$

com $\delta_N (p, i, Z) = \{(p, ZZ)\}$
$\delta_N (p, e, Z) = \{(p, \varepsilon)\}$

A partir de P_N ,vamos construir a Gramática Independente do Contexto equivalente.

Como P_N só possui <u>um</u> estado e <u>um</u> único símbolo de Pilha, o problema é simples. A gramática pretendida só terá duas variáveis,

S e [pZp].

e as produções serão:

S	$\rightarrow$ [pZp]	(porque só existe um estado)
[pZp]	$\rightarrow$ i [pZp] [pZp]	(porque $(p, ZZ) \in \delta_N (p, i, Z)$) {para <u>n</u> estados, seriam <u>n^2</u> produções}
[pZp]	$\rightarrow$ e	(porque $(p, \varepsilon) \in \delta_N (p, e, Z)$)

Substituindo [pZp] por A temos,

$S \rightarrow A$
$A \rightarrow iAA \mid e$

e, como S e A geram exactamente as mesma palavras, A pode ser eliminado e temos finalmente a GIC pretendida:

$G = (\{S\}, \{i, e\}, \{S \rightarrow iSS \mid e\}, S)$

Construção da Gramática Independente do Contexto:

Dado $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \{\})$,

definimos $G = (V, \Sigma, R, S)$, onde:

- $V = \{ [pXq] \mid p, q \in Q, X \in \Gamma \} \cup \{ S \}$
- $R = \{ S \rightarrow [q_0 Z_0 p] \mid p \in Q \} \cup$

$\{[pXp_{k+1}] \rightarrow a\ [p_1 Y_1 p_2]\ [p_2 Y_2 p_3] \ldots [p_k Y_k p_{k+1}] \mid$

$a \in \Sigma \cup \{\varepsilon\},$

$p_2, p_3, \ldots, p_{k+1} \in Q,$

$(p_1, Y_1 Y_2 \ldots Y_k) \in \delta(p, a, X)\}$

O Autómato Determinista de Pilha (ADP):

{ Não existem escolhas possíveis }

Definição: Um Autómato de Pilha $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

diz-se **Determinista** se e só se, $\forall\, q \in Q, \forall\, Z \in \Gamma$:

- Para todo o $a \in \Sigma \cup \{\varepsilon\}$ o conjunto $\delta(q, a, Z)$ tem, quando muito, um elemento;
- Se $\delta(q, \varepsilon, Z) \neq \emptyset$ então $\delta(q, a, Z) = \emptyset$, $\forall\, a \in \Sigma$.

Exemplo: O AP reconhecedor de $\{0^n 1^n \mid n \geq 0\}$ (pág. 17) é um ADP,

$$\begin{aligned}
\delta(q_0, 0, Z_0) &= \{(q_1, 0Z_0)\} \\
\delta(q_1, 0, 0) &= \{(q_1, 00)\} \\
\delta(q_1, 1, 0) &= \{(q_2, \varepsilon)\} \\
\delta(q_2, 1, 0) &= \{(q_2, \varepsilon)\} \\
\delta(q_2, \varepsilon, Z_0) &= \{(q_f, \varepsilon)\}
\end{aligned}$$

pois, todas as transições por leitura de símbolos são unívocas e a transição por ε não tem alternativa, por leitura de símbolo.

Exercício: Mostre que a linguagem $\{ w\, c\, w^R \mid w \in \{a, b\}^* \}$ é reconhecível por um ADP.

- Contudo, para a linguagem $\{ w\, w^R \mid w \in \{a, b\}^* \}$ (prova-se que) **não existe** ADP reconhecedor.