Formas Normais
Eliminação de ambiguidade

Flávio Márcio

Como eliminar a ambiguidade?

- Impor restrições às regras da GLC sem afetar a linguagem produzida pela mesma.
  - Através de alterações feitas pelo projetista da gramática e da linguagem. Utilizando sua experiência e insight.
  - Transformação para uma "Forma Normal" adequada. Através de algoritmos próprios.
- Vamos aprender a segunda que irá nos ajudar a desenvolver a experiência e insight; exigidos na primeira opção.

Formas Normais

- GLCs são muito flexíveis.
- Não existem restrições na forma do lado direito das regras.
- Isso facilita construção das gramáticas.
- Mas dificulta construção analisadores sintáticos (parsers).
- Formas normais:
  - Impõem restrições no lado direito das regras de GLCs.
  - Mas não reduzem o poder de expressão de GLCs.
  - Geradas automaticamente (via algoritmo).
- Exemplos de formas normais:
  - Forma Normal de Chomsky
  - Forma Normal de Greibach
  - Forma Normal de Backus-Nahur

Transformações de Gramáticas

- Transformação de uma GLC em uma forma normal:
  - Adição, modificação e eliminação de regras.
- Primeira transformação:
  - Símbolo inicial (S) deve se limitar a iniciar derivações
  - Isto é, S não deve ser uma variável recursiva.
  - Não deve ser possível ter $S \Rightarrow^{*} uSv$
- Suponha G= (V, Σ, P, S) uma GLC onde S é recursivo
- Então G'= (V $\cup$ {S'}, Σ, P $\cup$ {S' $\rightarrow$ S}, S') é uma GLC onde:
  - L(G') = L(G)
  - Símbolo inicial S' de G' não é mais recursivo.

Exemplo

G: $\mathbf{S} \rightarrow a\mathbf{S} \mid AB \mid AC$

$A \rightarrow aA \mid \lambda$

$B \rightarrow bB \mid bS$

$C \rightarrow cC \mid \lambda$

G': $S' \rightarrow S$

$S \rightarrow aS \mid AB \mid AC$

$A \rightarrow aA \mid \lambda$

$B \rightarrow bB \mid bS$

$C \rightarrow cC \mid \lambda$

Eliminação de Regras λ

- Variável anulável: variável que pode derivar λ.
- Se A é anulável, então: $A \Rightarrow^* \lambda$
- Gramática não-contrátil: não possui variáveis anuláveis.
  - Derivações não diminuem tamanho da forma sentencial.
- Gramática essencialmente não-contrátil:
  - Gramática não-contrátil
  - Mas admite-se uma única regra λ: $S \rightarrow \lambda$.
  - Todas derivações são não-contráteis, exceto $S \Rightarrow \lambda$.
- Exemplo:

$S \rightarrow aAb$
$A \rightarrow aA \mid B$
$B \rightarrow bB \mid \lambda$

Variáveis anuláveis: B e A

Eliminação de Regras λ

- Algoritmo para determinação de variáveis anuláveis:

  Entrada: G= (V, Σ, P, S)

  NULL= { A | A $\rightarrow \lambda \in$ P }

  repita

      PREV= NULL

      para cada variável A $\in$ V faça

          se existe regra A $\rightarrow$ w $\in$ P e w $\in$ PREV*

              então NULL= NULL U { A }

  até NULL == PREV

- Esse algoritmo segue uma abordagem “bottom-up” para calcular variáveis anuláveis.

Exemplo

- G: $S \rightarrow ACA$

  $A \rightarrow aAa \mid B \mid C$

  $B \rightarrow bB \mid b$

  $C \rightarrow cC \mid \lambda$
- Executando algoritmo:
  - NULL= { C }
  - PREV= { C } , NULL= { C, A }
  - PREV= { C, A }, NULL= { S, C, A }
  - PREV= { S, C, A }, NULL= { S, C, A }
- Como S é anulável, $\lambda \in L(G)$

Eliminação de Regras λ

- Técnica para eliminação de regras λ:
  - Calcular conjunto de variáveis anuláveis.
  - Eliminar regras que levam diretamente a λ.
  - Se $\lambda \in L(G)$, a regra $S \to \lambda$ deve estar presente.
  - Adicionar regras na gramática onde a ocorrência de variáveis anuláveis é omitida.
    - Exemplo: GLC com a seguinte regra: $A \to BABa$
    - Suponha que B é anulável
    - Logo, as seguintes regras devem ser adicionadas:

      $A \to ABa$ $\quad$ $A \to BAa$ $\quad$ $A \to Aa$

Exemplo

- G: $S \rightarrow ACA$

  $A \rightarrow aAa \mid B \mid C$

  $B \rightarrow bB \mid b$

  $C \rightarrow cC \mid \lambda$
- Variáveis anuláveis: S, C e A
- Gramática essencialmente não-contrátil equivalente a G:

  G_L: $S \rightarrow ACA \mid CA \mid AA \mid AC \mid A \mid C \mid \lambda$

  $A \rightarrow aAa \mid aa \mid B \mid C$

  $B \rightarrow bB \mid b$

  $C \rightarrow cC \mid c$

Exemplo

- $L(G)= a^* b^* c^*$
- G: $S \rightarrow ABC$

  $A \rightarrow aA \mid \lambda$

  $B \rightarrow bB \mid \lambda$

  $C \rightarrow cC \mid \lambda$
- Variáveis anuláveis: S, A, B e C
- Gramática essencialmente não-contrátil equivalente a G:

  G_L: $S \rightarrow ABC \mid AB \mid BC \mid AC \mid A \mid B \mid C \mid \lambda$

  $A \rightarrow aA \mid a$

  $B \rightarrow bB \mid b$

  $C \rightarrow cC \mid c$

Eliminação de Regras de Cadeia

- Uma regra da forma $A \rightarrow B$
  - Não aumenta tamanho da forma sentencial.
  - Não gera símbolos terminais.
  - Apenas renomeia uma variável.
- Tais regras são chamadas regras de cadeia (*chain rules*).
- Suponha:
  - $A \rightarrow aA \mid a \mid B$
  - $B \rightarrow bB \mid b \mid C$
- Eliminando cadeia $A \rightarrow B$:
  - $A \rightarrow aA \mid a \mid \mathbf{bB \mid b \mid C}$
  - $B \rightarrow bB \mid b \mid C$
- Infelizmente, uma nova cadeia apareceu: $A \rightarrow C$
- Ou seja, o procedimento deve ser reaplicado

Algoritmo CHAIN

- Algoritmo para cálculo de CHAIN(A): variáveis deriváveis a partir de A aplicando-se apenas regras de cadeia.
- Entrada: G= (V, Σ, P, S) , essencialmente não-contrátil.

CHAIN(A) = { A }
PREV= ϕ
repita
    NEW= CHAIN(A) – PREV
    PREV= CHAIN(A)
    para cada B $\in$ NEW faça
        para cada regra B $\rightarrow$ C faça
            CHAIN(A)= CHAIN(A) $\cup$ { C }
até CHAIN(A) == PREV

Exemplo

- G_L: $S \rightarrow ACA \mid CA \mid AA \mid AC \mid A \mid C \mid \lambda$

  $A \rightarrow aAa \mid aa \mid B \mid C$

  $B \rightarrow bB \mid b$

  $C \rightarrow cC \mid c$
- CHAIN(S):
  - CHAIN(S)= { S }, PREV= ϕ, NEW= { S }, PREV= { S }
  - CHAIN(S)= { S, A, C }
  - NEW= { A,C }, PREV= { S, A, C }
  - CHAIN(S)= { S, A, C, B }
  - NEW= { B }, PREV= { S, A, C, B }
  - CHAIN(S)= { S, A, C, B }
- CHAIN(A)= { A, B, C }; CHAIN(B)= { B }; CHAIN(C)= { C }

Exemplo

- G_L: $S \rightarrow ACA \mid CA \mid AA \mid AC \mid A \mid C \mid \lambda$

  $A \rightarrow aAa \mid aa \mid B \mid C$

  $B \rightarrow bB \mid b$

  $C \rightarrow cC \mid c$
- CHAIN(S)= { S, A, C, B }
- CHAIN(A)= { A, B, C } ; CHAIN(B)= { B } ; CHAIN(C)= { C }
- Gramática sem regras de cadeia:

  G_L: $S \rightarrow ACA \mid CA \mid AA \mid AC \mid aAa \mid aa \mid bB \mid b \mid cC \mid c \mid \lambda$

  $A \rightarrow aAa \mid aa \mid bB \mid b \mid cC \mid c$

  $B \rightarrow bB \mid b$

  $C \rightarrow cC \mid c$
- Novas regras: $A \rightarrow w$, onde $B \in$ CHAIN(A), $B \rightarrow w \in P$, $w \notin V$

Resumo

- Regras de GLC essencialmente não-contráteis e sem regras de cadeia tem uma das seguintes formas:
  - $S \rightarrow \lambda$
  - $A \rightarrow a$
  - $A \rightarrow w$, onde $w \in (V \cup \Sigma)^+$ e $|w|>=2$ possui tamanho pelo menos dois

Símbolos Inúteis

- Variável inútil: não aparece em derivações que geram strings; não contribui para geração de strings
- Exemplo:
- G : $S \rightarrow AC \mid BS \mid B$

$$\begin{array}{ll} A \rightarrow aA \mid aF & D \rightarrow aD \mid BD \mid C \\ B \rightarrow CF \mid b & E \rightarrow aA \mid BSA \\ C \rightarrow cC \mid D & F \rightarrow bB \mid b \end{array}$$

- L(G)= ?
- Um símbolo $x \in (V \cup \Sigma)$ é útil se: $S \Rightarrow^* uxv \Rightarrow^* w$, onde $u, v \in (V \cup \Sigma)^*$ e $w \in \Sigma^*$
- Um terminal é útil quando ocorre em um string $w \in L(G)$
- Uma variável A é útil se existe: $S \Rightarrow^* uAv \Rightarrow^* w$, $w \in L(G)$

Algoritmo TERM

- Algoritmo que calcula TERM: conjunto das variáveis que derivam strings
- Entrada: G= (V, Σ, P, S)

  TERM= { A | existe uma regra $A \rightarrow w \in$ P, com $w \in \Sigma^*$ }

  repita

  PREV= TERM

  para cada variável $A \in$ V faça

  se existe $A \rightarrow w \in$ P e $w \in$ (PREV U Σ)* então TERM = TERM U { A }

  até PREV == TERM
- Ao término do algoritmo, variáveis não pertencentes a TERM são inúteis

Exemplo

- G: $S \rightarrow AC \mid BS \mid B$

  $A \rightarrow aA \mid aF$ $\quad$ $D \rightarrow aD \mid BD \mid C$

  $B \rightarrow CF \mid b$ $\quad$ $E \rightarrow aA \mid BSA$

  $C \rightarrow cC \mid D$ $\quad$ $F \rightarrow bB \mid b$
- TERM= { B, F }
- PREV= { B, F }, TERM= { B, F, A, S }
- PREV= { B, F, A, S }, TERM= { B, F, A, S, E }
- PREV= { B, F, A, S, E }, TERM= { B, F, A, S, E }
- Variáveis inúteis: { C, D }
- G_T: $S \rightarrow BS \mid B$

  $A \rightarrow aA \mid aF$ $\quad$ $E \rightarrow aA \mid BSA$

  $B \rightarrow b$ $\quad$ $F \rightarrow bB \mid b$

Algoritmo REACH

- Algoritmo que calcula REACH: conjunto das variáveis alcançáveis a partir de S (isto é, $S \Rightarrow^* uVx$)
- Entrada: $G= (V, \Sigma, P, S)$

REACH= { S }

PREV= ϕ

repita

NEW= REACH – PREV

PREV= REACH

para cada $A \in$ NEW faça

para cada regra $A \rightarrow w$ faça

Adicione todas as variáveis de w em REACH

até REACH == PREV

Exemplo

- G_T: $S \rightarrow BS \mid B$

  $A \rightarrow aA \mid aF$ $\qquad$ $E \rightarrow aA \mid BSA$

  $B \rightarrow b$ $\qquad$ $F \rightarrow bB \mid b$
- REACH= { S }, PREV= ϕ
- NEW= { S }, PREV= { S }, REACH= { S, B }
- NEW= { B }, PREV= { S,B }, REACH= { S, B }
- Variáveis inalcançáveis a partir de S: A, E, F
- G_U: $S \rightarrow BS \mid B$

  $B \rightarrow b$
- Logo, L(G)= b^+

Exemplo

- Para remoção de variáveis inúteis:
  - Aplicar TERM e depois REACH (nesta ordem)
- G: $S \rightarrow a \mid AB$

  $A \rightarrow b$
- TERM= { S, A }
- G_T: $S \rightarrow a$

  $A \rightarrow b$
- REACH= { S }
- G_U: $S \rightarrow a$

- Se algoritmos forem aplicados na ordem inversa, o resultado seria incorreto

Forma Normal de Chomsky

- Uma GLC G= (V, Σ, P, S) está na Forma Normal de Chomsky se suas regras têm uma das seguintes formas:
  - $A \to BC$ onde $B,C \in V - \{ S \}$
  - $A \to a$
  - $S \to \lambda$
- Propriedade: árvores de derivação são sempre binárias
- Não é complexo converter uma gramática para a forma de Chomsky quando a mesma:
  - Não possui símbolo inicial recursivo
  - É essencialmente não-contrátil
  - Não possui regras de cadeia
  - Não possui símbolos inúteis

Forma Normal de Chomsky

- Exemplo: $A \rightarrow bDcF$
- Colocando na Forma de Chomsky:
  - Primeira transformação:

    $A \rightarrow B'DC'F$

    $B' \rightarrow b$

    $C' \rightarrow c$
  - Segunda transformação:

    $A \rightarrow B'T_1$

    $B' \rightarrow b$

    $C' \rightarrow c$

    $T_1 \rightarrow D\ T_2$

    $T_2 \rightarrow C'F$

Exemplo

- G: $S \rightarrow aABC \mid a$

  $A \rightarrow aA \mid a$

  $B \rightarrow bcB \mid bc$

  $C \rightarrow cC \mid c$
- Já satisfaz pré-condições: S não recursivo,essencialmente não-contrátil, sem regras de cadeia, sem símbolos inúteis
- G': $S \rightarrow A'\ T_1 \mid a$

  $A' \rightarrow a$

  $T_1 \rightarrow AT_2$

  $T_2 \rightarrow BC$

  $A \rightarrow A'A \mid a$

  $B \rightarrow B'\ T_3 \mid B'C'$

  $T_3 \rightarrow C'B$

  $C \rightarrow C'\ C \mid c$

  $B' \rightarrow b$

  $C' \rightarrow c$

Exemplo

- Suponha a gramática

  $S \rightarrow aXb \mid ab$

  $X \rightarrow aXb \mid ab$

- Forma Normal de Chomsky:

  $S \rightarrow AT \mid AB$

  $T \rightarrow XB$

  $X \rightarrow AT \mid AB$

  $A \rightarrow a$

  $B \rightarrow b$

Remoção de Recursão à Esquerda

- Recursão direta à esquerda pode produzir “loops infinitos em analisadores sintáticos top-down.
- Exemplo: $S \rightarrow Aa \quad A \rightarrow Aa \mid b$
  - $S \Rightarrow Aa \Rightarrow Aaa \Rightarrow Aaaa \Rightarrow ...$
- Suponha a regra genérica diretamente recursiva à esq.:
  - $A \rightarrow Au_1 \mid Au_2 \mid ... \mid Au_j \mid v_1 \mid v_2 \mid ... \mid v_k$
- Regra equivalente não-recursiva à esquerda:
  - $A \rightarrow v_1 \mid v_2 \mid ... \mid v_k \mid v_1Z \mid v_2Z \mid ... \mid v_kZ$
  - $Z \rightarrow u_1 Z \mid u_2 Z \mid ... \mid u_j Z \mid u_1 \mid u_2 \mid ... \mid u_j$

Remoção de Recursão à Esquerda

- Exemplo 1:

$A \rightarrow Aa \mid b$

$\Downarrow$

$A \rightarrow bZ \mid b$

$Z \rightarrow aZ \mid a$

- Exemplo 2:

$A \rightarrow Aa \mid Ab \mid b \mid c$

$\Downarrow$

$A \rightarrow bZ \mid cZ \mid b \mid c$

$Z \rightarrow aZ \mid bZ \mid a \mid b$

Remoção de Recursão à Esquerda

- Exemplo 3:

  $A \to AB \mid BA \mid a$

  $B \to b \mid c$

  $\Downarrow$

  $A \to BAZ \mid aZ \mid BA \mid a$

  $Z \to BZ \mid B$

  $B \to b \mid c$

- Exemplo 4:

  $A \to Aa \mid Aab \mid bb \mid b$ -- gera $(b \cup bb)(a \cup ab)^*$

  $\Downarrow$

  $A \to bb \mid b \mid bbZ \mid bZ$ -- gera $(b \cup bb)\ (Z \cup \lambda)$

  $Z \to aZ \mid abZ \mid a \mid ab$ -- gera $(a \cup ab)^*$

Remoção de Recursão à Esquerda

- Problema: recursão à esquerda indireta

  $A \rightarrow Bu$

  $B \rightarrow Av$

- Recursão indireta também pode gerar "loops infinitos"
- Solução: Forma Normal de Greibach

Forma Normal de Greibach

- Uma GLC G= (V, Σ, P, S) está na Forma Normal de Greibach se suas regras têm uma das seguintes formas:
  - $A \rightarrow aA_1A_2A_3 \ldots A_n$
  - $A \rightarrow a$
  - $S \rightarrow \lambda$

  onde $a \in \Sigma$ e $A_i \in V - \{ S \}$, para i= 1, 2, ..., n
- Primeiro passo:
  - Transformar para forma normal de Chomsky
  - E eliminar recursão direta à esquerda
- Segundo passo:
  - Ordenar variáveis; cada variável ganha um número
  - S=1, A= 2, B= 3, etc

Forma Normal de Greibach

- Terceiro passo: regras das seguintes formas
  - $S \to \lambda$
  - $A \to aw$
  - $A \to Bw$

  onde $w \in V^*$ e número(B) > número(A)

Exemplo

- G: $S \rightarrow AB \mid \lambda$

  $A \rightarrow AB \mid CB \mid a$

  $B \rightarrow AB \mid b$

  $C \rightarrow AC \mid c$
- Numerando variáveis:
  - S=1; A= 2; B= 3; C= 4
- Eliminando recursão à esquerda em A:

  G: $S \rightarrow AB \mid \lambda$

  $A \rightarrow CBR_1 \mid aR_1 \mid CB \mid a$

  $B \rightarrow AB \mid b$

  $C \rightarrow AC \mid c$

  $R_1 \rightarrow BR_1 \mid B$

Exemplo

- Gramática até o momento:

  G: $S \rightarrow AB \mid \lambda$ -- ok, A > S
  $A \rightarrow CBR_1 \mid aR_1 \mid CB \mid a$ -- ok, C > A
  $B \rightarrow AB \mid b$ -- não ok
  $C \rightarrow AC \mid c$ -- não ok
  $R_1 \rightarrow BR_1 \mid B$

- Movendo lado direito de A para regra $B \rightarrow AB$ e $C \rightarrow AC$

  G: $S \rightarrow AB \mid \lambda$
  $A \rightarrow CBR_1 \mid aR_1 \mid CB \mid a$
  $B \rightarrow CBR_1B \mid aR_1B \mid CBB \mid aB \mid b$ -- ok, C > B
  $C \rightarrow CBR_1C \mid aR_1C \mid CBC \mid aC \mid c$ -- recursiva esq.
  $R_1 \rightarrow BR_1 \mid B$

Exemplo

- Gramática até o momento:

  G: $S \rightarrow AB \mid \lambda$

  $A \rightarrow CBR_1 \mid aR_1 \mid CB \mid a$

  $B \rightarrow CBR_1B \mid aR_1B \mid CBB \mid aB \mid b$

  $C \rightarrow CBR_1C \mid aR_1C \mid CBC \mid aC \mid c$ -- recursiva esq.

  $R_1 \rightarrow BR_1 \mid B$

- Removendo recursão à esquerda na regra C:

  G: $S \rightarrow AB \mid \lambda$

  $A \rightarrow CBR_1 \mid aR_1 \mid CB \mid a$

  $B \rightarrow CBR_1B \mid aR_1B \mid CBB \mid aB \mid b$

  $C \rightarrow aR_1C \mid aC \mid c \mid aR_1CR_2 \mid aCR_2 \mid cR_2$

  $R_1 \rightarrow BR_1 \mid B$

  $R_2 \rightarrow BR_1C\,R_2 \mid BC\,R_2 \mid BR_1C \mid BC$

Exemplo

- Gramática até o momento (sem R_1 e R_2):

  G: $S \rightarrow AB \mid \lambda$ -- ok 3º passo

  $A \rightarrow CBR_1 \mid aR_1 \mid CB \mid a$ -- ok 3º passo

  $B \rightarrow CBR_1B \mid aR_1B \mid CBB \mid aB \mid b$ -- ok 3º passo

  $C \rightarrow aR_1C \mid aC \mid c \mid aR_1CR_2 \mid aCR_2 \mid cR_2$ -- ok 3º passo

- Regra C já está de acordo com Greibach
- Movendo regra C para lado direito de B:

  $B \rightarrow aR_1B \mid aB \mid b$

  $\rightarrow aR_1C\mathbf{BR_1B} \mid aC\mathbf{BR_1B} \mid c\mathbf{BR_1B} \mid aR_1CR_2\mathbf{BR_1B} \mid$

  $aCR_2\mathbf{BR_1B} \mid cR_2\mathbf{BR_1B}$

  $\rightarrow aR_1C\mathbf{BB} \mid aC\mathbf{BB} \mid c\mathbf{BB} \mid aR_1CR_2\mathbf{BB} \mid aCR_2\mathbf{BB} \mid$

  $cR_2\mathbf{BB}$ -- agora B está de acordo com Greibach

- Repetir procedimento para A, S, R_1 e R_2 (nesta ordem)

Greibach: Comentários Finais

- Forma Normal de Greibach:
  - Gramáticas grandes (com diversas regras)
  - Gramáticas sem a simplicidade original
- Transformação de uma GLC para Greibach pode ser automatizada (via algoritmo descrito nos slides anteriores)
- No entanto, gramáticas na Forma Normal de Greibach possuem propriedades interessantes:
  - Toda derivação adiciona um terminal no início da string que está sendo derivada
  - Logo, não existe recursão à esquerda (direta ou indireta)
  - Strings de tamanho n > 0 são derivadas aplicando-se exatamente n regras da gramática

Fatoração à esquerda

- Reescrever a gramática adiando a escolha da substituição de um não terminal pelo lado direito da produção.
- É uma transformação que prepara a gramática para um predictive parsing.
- Se A -> $ab_{\mathbf{1}}$ | $ab_{\mathbf{2}}$ não sabemos que produção utilizar podemos adiar a decisão fazendo:
- A -> aR
- R -> $b_{\mathbf{1}}$ | $b_{\mathbf{2}}$

Fatoração à esquerda

- 1. Para cada não terminal *A* encontre o maior prefixo "a" comum a duas ou mais alternativas.
- 2. Se "a" <> "lambida" substitua todas as produções:
- A -> $ab_{\mathbf{1}}$ | $ab_{\mathbf{2}}$ | ... | $ab_{\mathbf{n}}$
- Por
- A -> aR
- R -> $b_{\mathbf{1}}$ | $b_{\mathbf{2}}$ | ... | $b_{\mathbf{n}}$
- 3. Repita o processo até que não sobre duas alternativas com um prefixo comum.

GLC e Linguagens de Programação

- Notação BNF (Backus-Nahur Form): notação normalmente usada para definir GLCs de linguagens de programação
- Propostas quando da definição da gramática de Algol 60
- Notação BNF:
  - Não se usa λ
  - Terminais: negrito (ou começando com minúsculas)
  - Variáveis: < .... > (ou começando com maiúsculas)
  - { A } denota zero ou mais repetições de A
  - [A] denota que A é opcional
- BNF de Java:
  - The Java Language Specification
  - http://java.sun.com/docs/books/jls/

BNF de Java

```
CompilationUnit: [ package QualifiedIdentifier ; ] { ImportDeclaration }
     { TypeDeclaration }

ImportDeclaration: import Identifier { . Identifier } [ . * ] ;

TypeDeclaration: ClassOrInterfaceDeclaration ;

ClassOrInterfaceDeclaration: ModifiersOpt (ClassDeclaration | InterfaceDeclaration)

ClassDeclaration: class Identifier [extends Type] [implements TypeList] ClassBody

InterfaceDeclaration: interface Identifier [extends TypeList] InterfaceBody

TypeList: Type { , Type }

ClassBody: { {ClassBodyDeclaration} }

InterfaceBody: { {InterfaceBodyDeclaration} }

ClassBodyDeclaration:
    ;
    [static] Block
    ModifiersOpt MemberDecl
```

BNF de Java

```
MemberDecl:
    MethodOrFieldDecl
    void Identifier MethodDeclaratorRest
    Identifier ConstructorDeclaratorRest
    ClassOrInterfaceDeclaration

MethodOrFieldDecl: Type Identifier MethodOrFieldRest

MethodOrFieldRest:
    VariableDeclaratorRest
    MethodDeclaratorRest

InterfaceBodyDeclaration:
    ;
    ModifiersOpt InterfaceMemberDecl

InterfaceMemberDecl:
    InterfaceMethodOrFieldDecl
    void Identifier VoidInterfaceMethodDeclaratorRest
    ClassOrInterfaceDeclaration

InterfaceMethodOrFieldDecl: Type Identifier InterfaceMethodOrFieldRest
```

BNF de Java

```
InterfaceMethodOrFieldRest:
    ConstantDeclaratorsRest ;
    InterfaceMethodDeclaratorRest

MethodDeclaratorRest:
    FormalParameters BracketsOpt [throws QualifiedIdentifierList]( MethodBody | ; )

VoidMethodDeclaratorRest:
    FormalParameters [throws QualifiedIdentifierList] ( MethodBody | ;  )

InterfaceMethodDeclaratorRest:
    FormalParameters BracketsOpt [throws QualifiedIdentifierList] ;

VoidInterfaceMethodDeclaratorRest:
    FormalParameters [throws QualifiedIdentifierList] ;

ConstructorDeclaratorRest:
      FormalParameters [throws QualifiedIdentifierList] MethodBody

QualifiedIdentifierList: QualifiedIdentifier { , QualifiedIdentifier }

FormalParameters: ( [FormalParameter { , FormalParameter}] )
```

For compatibility with older versions, a declaration form for a method that returns an array is allowed to place (some or all of) the empty bracket pairs that form the declaration of the array type after the parameter list.

BNF de Java

```
FormalParameter:
    [final] Type VariableDeclaratorId

MethodBody:
    Block

ModifiersOpt:
    { Modifier }

Modifier:
    public
    protected
    private
    static
    abstract
    final
    native
    synchronized
    transient
    volatile
    strictfp
```