

3

Expressões Regulares e PEGs

Este capítulo apresenta uma formalização de expressões regulares usando semântica natural, discute a correspondência entre expressões regulares e PEGs, e define uma transformação entre expressões regulares e PEGs equivalentes. Além disso, mostramos aqui como modificar uma expressão regular de modo a obter uma expressão sem subexpressões da forma e_1^* onde e_1 casa a cadeia vazia. Essa modificação nos permite transformar expressões regulares em PEGs sem produções recursivas à esquerda.

Na próxima seção revisamos alguns conceitos de expressões regulares e na seção 3.2 apresentamos nossa formalização de expressões regulares baseada em semântica natural. Na seção 3.3, discutimos a equivalência entre expressões regulares e PEGs. A seção 3.4 descreve como podemos transformar uma expressão regular em uma PEG equivalente. A seção 3.5 mostra como podemos reescrever expressões regulares da forma e_1^* onde e_1 casa a cadeia vazia. Por fim, na seção 3.6, provamos a equivalência entre expressões regulares e PEGs obtidas a partir da transformação apresentada na seção 3.4.

3.1

Expressões Regulares

Dado um alfabeto finito Σ , podemos definir uma expressão regular e_0 indutivamente como a seguir, onde $a \in \Sigma$, e e_1 e e_2 também são expressões regulares:

$$e_0 = \emptyset \mid \varepsilon \mid a \mid e_1 e_2 \mid e_1 | e_2 \mid e_1^*$$

Geralmente a linguagem definida por uma expressão regular e_0 , denotada por $L(e_0)$, é especificada através de operações sobre conjuntos [Hopcroft e Ullman, 1979, Sipser, 1996], conforme a tabela 3.1.

A linguagem definida pela expressão regular \emptyset é o conjunto vazio. Quando não queremos definir uma expressão regular cuja linguagem é vazia o uso de \emptyset é desnecessário, uma vez que qualquer expressão regular e_0 pode ser reduzida a uma expressão regular e_1 que define a mesma linguagem, onde $e_1 = \emptyset$ ou e_1 não possui \emptyset como uma subexpressão. Essa transformação é

Expressão Regular	Linguagem Correspondente
\emptyset	\emptyset
ε	$\{\varepsilon\}$
a	$\{a\}$
$e_1 e_2$	$L(e_1) L(e_2)$
$e_1 e_2$	$L(e_1) \cup L(e_2)$
e_1^*	$L(e_1)^*$

Tabela 3.1: Expressões Regulares e suas Linguagens Correspondentes

baseada nas seguintes igualdades:

$$L(e_0 \emptyset) = L(\emptyset) \quad L(e_0 | \emptyset) = L(e_0) \quad L(\emptyset^*) = L(\varepsilon)$$

Como o uso da expressão regular \emptyset é bastante restrito, em algumas discussões não iremos considerar essa forma de expressão regular. Em particular, iremos assumir que \emptyset nunca é uma subexpressão de uma expressão regular $e_0 \neq \emptyset$.

3.2

Definição de Expressões Regulares Usando Semântica Natural

Como nosso objetivo principal é estabelecer a correspondência entre expressões regulares e PEGs, apresentaremos uma formalização diferente de expressões regulares. Nessa nova formalização, usamos a relação $\overset{\text{RE}}{\rightsquigarrow}$ para dar significado a uma expressão regular.

Definimos $\overset{\text{RE}}{\rightsquigarrow}$ como uma relação $(e_0 \times \Sigma^*) \times \Sigma^*$, onde $\overset{\text{RE}}{\rightsquigarrow}$ relaciona uma expressão regular e_0 e uma entrada xy com um sufixo y da entrada. Usamos a notação $e_0 xy \overset{\text{RE}}{\rightsquigarrow} y$ para indicar que $((e_0, xy), y) \in \overset{\text{RE}}{\rightsquigarrow}$. A figura 3.1 apresenta a definição de $\overset{\text{RE}}{\rightsquigarrow}$ usando semântica natural.

A relação $\overset{\text{RE}}{\rightsquigarrow}$ não é uma função, uma vez que um par (e_0, x) pode não se relacionar com nenhum sufixo de x , ou se relacionar com diferentes sufixos. A seguir, discutimos as regras de $\overset{\text{RE}}{\rightsquigarrow}$.

A regra *empty.1* trata do caso em que a expressão regular representa a cadeia vazia. A regra *char.1* trata de expressões regulares da forma a , e a regra *con.1* trata de expressões regulares da forma $e_1 e_2$.

As regras *choice.1* e *choice.2* tratam de expressões regulares da forma $e_1 | e_2$. Usaremos o termo *escolha* ao nos referirmos à união das linguagens de duas expressões regulares. O resultado do casamento de uma escolha pode ser tanto o resultado do casamento de e_1 (regra *choice.1*), como o resultado do casamento de e_2 (regra *choice.2*).

Finalmente, as regras *rep.1* e *rep.2* tratam de expressões regulares da

$$\begin{array}{l}
 \text{Cadeia Vazia} \quad \frac{}{\varepsilon \ x \overset{\text{RE}}{\rightsquigarrow} x} \text{ (empty.1)} \qquad \text{Caractere} \quad \frac{}{a \ ax \overset{\text{RE}}{\rightsquigarrow} x} \text{ (char.1)} \\
 \\
 \text{Concatenação} \quad \frac{e_1 \ xyz \overset{\text{RE}}{\rightsquigarrow} yz \quad e_2 \ yz \overset{\text{RE}}{\rightsquigarrow} z}{e_1 e_2 \ xyz \overset{\text{RE}}{\rightsquigarrow} z} \text{ (con.1)} \\
 \\
 \text{Escolha} \quad \frac{e_1 \ xy \overset{\text{RE}}{\rightsquigarrow} y}{e_1 \mid e_2 \ xy \overset{\text{RE}}{\rightsquigarrow} y} \text{ (choice.1)} \qquad \frac{e_2 \ xy \overset{\text{RE}}{\rightsquigarrow} y}{e_1 \mid e_2 \ xy \overset{\text{RE}}{\rightsquigarrow} y} \text{ (choice.2)} \\
 \\
 \text{Repetição} \quad \frac{}{e_0^* \ x \overset{\text{RE}}{\rightsquigarrow} x} \text{ (rep.1)} \quad \frac{e_0 \ xyz \overset{\text{RE}}{\rightsquigarrow} yz \quad e_0^* \ yz \overset{\text{RE}}{\rightsquigarrow} z}{e_0^* \ xyz \overset{\text{RE}}{\rightsquigarrow} z}, \ x \neq \varepsilon \text{ (rep.2)}
 \end{array}$$

Figura 3.1: Definição da Relação $\overset{\text{RE}}{\rightsquigarrow}$ Usando Semântica Natural

forma e_0^* . Usaremos o termo *repetição* ao nos referirmos ao fecho de Kleene de uma expressão regular. A regra *rep.1* trata do caso em que nenhum caractere da entrada é consumido, enquanto a regra *rep.2* trata do caso em que algum caractere da entrada é consumido. Caso a regra *rep.2* não tivesse a condição $x \neq \varepsilon$, poderíamos casar a cadeia vazia de modo não determinístico através das regras *rep.1* e *rep.2*.

Podemos notar que nenhuma regra de $\overset{\text{RE}}{\rightsquigarrow}$ trata da expressão regular \emptyset , de modo que essa expressão não se relaciona com nenhuma cadeia, e portanto a linguagem que ela define é vazia.

3.2.1

Correspondência com a Definição Usual de Expressões Regulares

Dada a nova definição de expressões regulares baseada na relação $\overset{\text{RE}}{\rightsquigarrow}$, vamos estabelecer a sua correspondência com a definição usual de expressões regulares que é baseada em operações sobre conjuntos. Para isso, vamos precisar do seguinte lema:

Lema 3.2.1. *Dada uma expressão regular e_0 e uma cadeia x , temos que $x \in L(e_0)^*$ se e somente se $x = \varepsilon$ ou $x = x_1x_2$, onde $x_1 \in L(e_0)$, $x_2 \in L(e_0)^*$ e $x_1 \neq \varepsilon$.*

Demonstração. Por definição, temos que $L(e_0)^* = \bigcup_{i=0}^{\infty} L(e_0)^i$. Portanto, $x \in L(e_0)^* \Leftrightarrow \exists i \cdot x \in L(e_0)^i$. Seja j o menor número natural tal que $x \in L(e_0)^j$, temos então dois casos dependendo se j é ou não maior que zero.

Se $j = 0$, temos que $L(e_0)^0 = \{\varepsilon\}$, e portanto $x = \varepsilon$.

Se $j > 0$, temos que $L(e_0)^j = L(e_0) L(e_0)^{j-1}$. Como $x \in L(e_0) L(e_0)^{j-1}$, então $x = x_1 x_2$, com $x_1 \in L(e_0)$ e $x_2 \in L(e_0)^{j-1}$. Nesse caso, a cadeia x_1 é diferente de ε , pois se $x_1 = \varepsilon$ então teríamos que $x = x_2 \in L(e_0)^{j-1}$, o que contradiz o fato de que escolhemos o menor natural j tal que $x \in L(e_0)^j$. \square

A proposição a seguir estabelece a correspondência entre a formalização usual de expressões regulares e a nossa formalização:

Proposição 3.2.2. *Dada uma expressão regular e_0 e uma cadeia x , para qualquer cadeia y temos que $x \in L(e_0)$ se e somente se $e_0 xy \xrightarrow{RE} y$.*

Demonstração. A prova das duas partes é por indução na complexidade do par (e_0, x) , que é dada pela estrutura de e_0 e pelo comprimento de x . Dados os pares (e_1, x_1) e (e_2, x_2) , temos que a complexidade do primeiro par é maior do que a do segundo se a estrutura de e_1 é maior do que a estrutura de e_2 , ou se a estrutura de e_1 é igual à estrutura de e_2 e $|x_1| > |x_2|$.

(\Rightarrow) : Quando $e_0 = \varepsilon$, somente $\varepsilon \in L(\varepsilon)$. Nesse caso, pela regra *empty.1* temos que $\varepsilon xy \xrightarrow{RE} y$, onde $x = \varepsilon$.

Quando $e_0 = a$, somente $a \in L(a)$. Nesse caso, pela regra *char.1* temos que $a ay \xrightarrow{RE} y$.

Quando $e_0 = e_1 e_2$, temos que $L(e_1 e_2) = L(e_1) L(e_2)$. Seja $x_1 x_2 \in L(e_1 e_2)$, onde $x_1 \in L(e_1)$ e $x_2 \in L(e_2)$. Pela hipótese de indução temos que $e_1 x_1 x_2 y \xrightarrow{RE} x_2 y$ e que $e_2 x_2 y \xrightarrow{RE} y$. Assim, pela regra *con.1* concluímos que $e_1 e_2 x_1 x_2 y \xrightarrow{RE} y$.

Quando $e_0 = e_1 | e_2$, temos que $L(e_1 | e_2) = L(e_1) \cup L(e_2)$. Vamos considerar dois casos: $x \in L(e_1)$ e $x \in L(e_2)$.

Se $x \in L(e_1)$, pela hipótese de indução temos que $e_1 xy \xrightarrow{RE} y$, e pela regra *choice.1* concluímos que $e_1 | e_2 xy \xrightarrow{RE} y$.

Se $x \in L(e_2)$, pela hipótese de indução temos que $e_2 xy \xrightarrow{RE} y$, e pela regra *choice.2* concluímos que $e_1 | e_2 xy \xrightarrow{RE} y$.

Finalmente, quando $e_0 = e_1^*$, seja $x \in L(e_1^*)$, pelo lema 3.2.1 sabemos que $x = \varepsilon$ ou $x = x_1 x_2$, onde $x_1 \in L(e_1)$, $x_2 \in L(e_1^*)$, e $x_1 \neq \varepsilon$.

Se $x = \varepsilon$, pela regra *rep.1* concluímos que $e_1^* y \xrightarrow{RE} y$.

Se $x = x_1 x_2$, dado que a estrutura de e_1 é menor do que a estrutura de e_1^* , pela hipótese de indução temos que $e_1 x_1 x_2 y \xrightarrow{RE} x_2 y$. Dado que $x_1 \neq \varepsilon$, sabemos que $|x_2| < |x_1 x_2|$. Assim, pela hipótese de indução temos que $e_1^* x_2 y \xrightarrow{RE} y$, e pela regra *rep.2* concluímos que $e_1^* x_1 x_2 y \xrightarrow{RE} y$.

(\Leftarrow) : Quando $e_0 = \varepsilon$, temos o casamento $\varepsilon xy \xrightarrow{RE} y$, e pela definição usual de expressões regulares temos que $\varepsilon \in L(\varepsilon)$.

Quando $e_0 = a$, temos o casamento $a ay \xrightarrow{RE} y$, e pela definição usual de expressões regulares temos que $a \in L(a)$.

Quando $e_0 = e_1 e_2$, temos o casamento $e_1 e_2 x_1 x_2 y \xrightarrow{\text{RE}} y$, onde pela regra *con.1* sabemos que $e_1 x_1 x_2 y \xrightarrow{\text{RE}} x_2 y$ e que $e_2 x_2 y \xrightarrow{\text{RE}} y$. Assim, pela hipótese de indução temos que $x_1 \in L(e_1)$ e que $x_2 \in L(e_2)$, e portanto $x_1 x_2 \in L(e_1 e_2)$.

Quando $e_0 = e_1 | e_2$, temos o casamento $e_1 | e_2 xy \xrightarrow{\text{RE}} y$. Vamos considerar dois casos: no primeiro caso a regra *choice.1* foi usada, e no segundo caso a regra *choice.2* foi usada.

Se *choice.1* foi usada, então $e_1 xy \xrightarrow{\text{RE}} y$, e pela hipótese de indução temos que $x \in L(e_1)$. Assim, concluímos que $x \in L(e_1 | e_2)$.

Se *choice.2* foi usada, então $e_2 xy \xrightarrow{\text{RE}} y$, e pela hipótese de indução temos que $x \in L(e_2)$. Assim, concluímos que $x \in L(e_1 | e_2)$.

Quando $e_0 = e_1^*$, temos o casamento $e_1^* xy \xrightarrow{\text{RE}} xy$. Vamos considerar dois casos: no primeiro caso a regra *rep.1* foi usada, e no segundo caso a regra *rep.2* foi usada.

Se *rep.1* foi usada, então $e_1^* xy \xrightarrow{\text{RE}} y$, onde $x = \varepsilon$, e pela definição usual de expressões regulares sabemos que $\varepsilon \in L(e_1^*)$.

Se *rep.2* foi usada, seja $x = x_1 x_2$, temos que $e_1 x_1 x_2 y \xrightarrow{\text{RE}} x_2 y$ e que $e_1^* x_2 y \xrightarrow{\text{RE}} y$, onde $x_1 \neq \varepsilon$. Dado que a estrutura de e_1 é menor do que a estrutura de e_1^* , pela hipótese de indução temos que $x_1 \in L(e_1)$, e como $|x_2| < |x_1 x_2|$ pela hipótese de indução temos que $x_2 \in L(e_1^*)$. Como $L(e_1 e_1^*) \subset L(e_1^*)$, concluímos que $x_1 x_2 \in L(e_1^*)$. \square

3.2.2 Propriedades de Expressões Regulares

A seguir, definimos um lema a respeito do casamento de expressões regulares quando mudamos o sufixo da entrada que não foi casado:

Lema 3.2.3. *Dada uma expressão regular e_0 , temos que se $e_0 xy \xrightarrow{\text{RE}} y$ então $\forall y' \cdot e_0 xy' \xrightarrow{\text{RE}} y'$.*

Demonstração. A prova é por indução na altura da árvore de prova dada por $\xrightarrow{\text{RE}}$.

No caso de uma repetição e_0^* , temos que as regras *rep.1* e *rep.2* podem ter sido usadas.

Se a regra *rep.1* foi usada, sabemos que $e_0^* xy \xrightarrow{\text{RE}} y$, onde $x = \varepsilon$, e pela própria regra *rep.1* concluímos que $e_0^* xy' \xrightarrow{\text{RE}} y'$.

Se a regra *rep.2* foi usada, sabemos que $e_0 x_1 x_2 y \xrightarrow{\text{RE}} x_2 y$ e que $e_0^* x_2 y \xrightarrow{\text{RE}} y$, onde $x = x_1 x_2$. Pela hipótese de indução temos que $e_0 x_1 x_2 y' \xrightarrow{\text{RE}} x_2 y'$ e que $e_0^* x_2 y' \xrightarrow{\text{RE}} y'$, e pela regra *rep.2* concluímos que $e_0^* x_1 x_2 y' \xrightarrow{\text{RE}} y'$.

A prova dos outros casos é similar. \square

No próximo capítulo, iremos mostrar um lema análogo para o casamento de CFGs e discutiremos porque no caso de PEGs não é possível definir tal lema.

3.3

Equivalência Entre Expressões Regulares e PEGs

Nesta seção vamos discutir a correspondência entre expressões regulares e PEGs, e definir quando uma expressão regular é equivalente a uma PEG.

Vamos usar ao longo do texto a notação $x' \preceq x$ para denotar que a cadeia x' é um sufixo da cadeia x , e a notação $x' \prec x$ para denotar que x' é um sufixo próprio de x .

A seguir, apresentamos a nossa definição de equivalência entre expressões regulares e PEGs:

Definição 3.3.1. *Dada uma PEG $G = (V, T, P, p_S)$ e uma expressão regular e_0 sobre o alfabeto $\Sigma = T$, dizemos que elas são equivalentes se para toda cadeia x temos que:*

1. $\forall y \preceq x \cdot G \ x \xrightarrow{PEG} y \Rightarrow e_0 \ x \xrightarrow{RE} y.$
2. $\forall y \preceq x \cdot e_0 \ x \xrightarrow{RE} y \Rightarrow \exists y' \preceq x \cdot G \ x \xrightarrow{PEG} y'.$

Pela definição anterior, uma PEG G é equivalente a uma expressão regular e_0 se para toda cadeia x a cadeia y resultante do casamento de G pode ser obtida através do casamento de e_0 , e se para toda cadeia x em que o casamento de e_0 é bem sucedido o casamento de G também é bem sucedido.

Dada uma cadeia x , uma expressão regular e_0 e uma PEG G , vamos usar a notação $G \xrightarrow{\preceq x} e_0$ para denotar que para todo $x' \preceq x$ temos que $G \ x' \xrightarrow{PEG} y \Rightarrow e_0 \ x' \xrightarrow{RE} y$, e a notação $G \xrightarrow{\prec x} e_0$ para denotar que para todo $x' \prec x$ temos que $G \ x' \xrightarrow{PEG} y \Rightarrow e_0 \ x' \xrightarrow{RE} y$.

De maneira análoga, dada uma cadeia x , uma expressão regular e_0 e uma PEG G , vamos usar a notação $G \xleftarrow{\preceq x} e_0$ para denotar que para todo $x' \preceq x$ temos que $e_0 \ x' \xrightarrow{RE} y \Rightarrow G \ x' \xrightarrow{PEG} y'$, e a notação $G \xleftarrow{\prec x} e_0$ para denotar que para todo $x' \prec x$ temos que $e_0 \ x' \xrightarrow{RE} y \Rightarrow G \ x' \xrightarrow{PEG} y'$.

Como uma expressão regular pode se relacionar com diferentes sufixos de uma cadeia x , pois o casamento usando \xrightarrow{RE} é não determinístico, e uma PEG pode se relacionar com apenas um único sufixo de uma cadeia x , pois o casamento usando \xrightarrow{PEG} é determinístico, quando uma PEG G é equivalente a uma expressão regular e_0 temos que a linguagem definida por e_0 pode ter cadeias que não pertencem à linguagem de G .

Vamos discutir alguns exemplos de expressões regulares e PEGs e analisar se elas são equivalentes ou não.

Se a expressão regular é da forma a , é trivial obter uma PEG equivalente cuja expressão de parsing inicial também é da forma a .

No caso da expressão regular $a|ab$, ela é equivalente a uma PEG cuja expressão de parsing inicial é a/ab , embora a expressão regular defina a linguagem $\{a, ab\}$, enquanto que a PEG define a linguagem $\{a\}$.

A expressão regular $a(b|bb)$ é equivalente a uma PEG cuja expressão de parsing inicial é $a(b/bb)$. Contudo, a expressão regular $(a|aa)b$ não é equivalente a uma PEG cuja expressão de parsing inicial é $(a/aa)b$, pois o casamento dessa expressão regular é bem sucedido para a entrada aab , ao passo que o casamento dessa PEG não.

Para obter uma PEG equivalente a uma expressão regular da forma $(e_1|e_2)e_3$, precisamos criar uma PEG na qual a expressão de parsing p_2 , equivalente a e_2 , pode casar quando o casamento da expressão de parsing p_1 , equivalente a e_1 , é bem sucedido, mas o casamento da expressão de parsing p_3 , equivalente a e_3 , falha. Podemos conseguir isso distribuindo a expressão regular e_3 entre as alternativas da escolha $e_1|e_2$, o que nos daria uma expressão de parsing da forma $p_1 p_3 / p_2 p_3$. Voltando ao exemplo anterior, temos que a expressão regular $(a|aa)b$ é equivalente a uma PEG cuja expressão de parsing inicial é ab/aab .

No caso da expressão regular b^*b , ela não é equivalente a uma PEG cuja expressão de parsing inicial é b^*b . Como o casamento usando $\overset{\text{RE}}{\rightsquigarrow}$ é não determinístico, dada a entrada bb , a expressão b^* pode casar ε , b ou bb , e portanto o casamento da expressão regular b^*b pode ser bem sucedido. Por outro lado, como um casamento usando $\overset{\text{PEG}}{\rightsquigarrow}$ é determinístico, dada a entrada bb temos que b^* sempre casa bb , e o resultado do casamento da expressão de parsing b^*b sempre é **fail**. Este problema é análogo ao discutido anteriormente, uma vez que expressões regulares da forma e^* podem ser reescritas como $e e^* | \varepsilon$.

Para obter uma PEG equivalente a uma expressão regular da forma $e_1^* e_2$, vamos reescrever essa expressão regular como a seguir:

$$e_1^* e_2 \equiv (e_1 e_1^* | \varepsilon) e_2 \equiv e_1 e_1^* e_2 | e_2$$

Como podemos ver, a expressão regular $e_1^* e_2$ aparece novamente depois de a reescrevermos. Em PEGs, podemos expressar definições recursivas através de não terminais, de modo que a PEG a seguir é equivalente à expressão regular $e_1 e_1^* e_2 | e_2$, onde p_1 é equivalente a e_1 , e p_2 é equivalente a e_2 :

$$A \rightarrow p_1 A / p_2$$

Quando o casamento da expressão regular anterior é bem sucedido através da *choice.1*, sabemos que o casamento da expressão de parsing $p_1 A$ é bem sucedido. Já quando o casamento da expressão regular anterior usa a regra *choice.2*, sabemos que o casamento da expressão de parsing p_2 é bem sucedido.

Seguindo nosso exemplo, a gramática a seguir é equivalente à expressão regular $b^* b$:

$$A \rightarrow bA / b$$

Como vimos anteriormente, essa gramática define uma repetição gulosa. Dada a PEG acima e a entrada bb , o primeiro b da entrada é casado pela subexpressão b da concatenação bA e em seguida temos o casamento do não terminal A , onde a entrada restante é b . Novamente, a subexpressão b da concatenação bA casa um b e em seguida temos o casamento de A , onde a entrada restante agora é ε . Como os casamentos de bA e de b falham para a entrada ε , o casamento de A falha para a entrada ε , e portanto o casamento da concatenação bA falha para a entrada b . Como o casamento da segunda alternativa da escolha associada ao não terminal A é bem sucedido para essa entrada, temos que o casamento de A é bem sucedido para a entrada b e que o casamento de bA é bem sucedido para a entrada bb . Assim, o não terminal A casa a entrada bb .

Como o casamento em PEGs é determinístico, dada a entrada bb , o não terminal A sempre casa bb , ao passo que a expressão regular $b^* b$, dada a mesma entrada, pode casar b ou bb .

3.4

Transformação de uma Expressão Regular em uma PEG Equivalente

Nesta seção vamos apresentar a função Π , que transforma uma dada expressão regular em uma PEG equivalente.

A função Π , cuja definição aparece na figura 3.2, recebe uma expressão regular e_0 e uma PEG $G_k = (V_k, T, P_k, p_k)$, que é equivalente a uma expressão regular e_k , e nos dá uma PEG que é equivalente à expressão regular $e_0 e_k$.

Podemos ver a gramática G_k como uma continuação, que determina como deve ser o casamento na PEG depois que o casamento da expressão de parsing equivalente a e_0 ocorre. Quando G_k é equivalente a um expressão regular $e_k = \varepsilon$, a transformação Π nos dá uma PEG equivalente à expressão regular $e_0 \varepsilon \equiv e_0$.

Se a expressão regular é a constante \emptyset , o resultado de $\Pi(\emptyset, G_k)$ é uma gramática cuja expressão de parsing inicial é $!\varepsilon$. O resultado do casamento

$$\begin{aligned}
\Pi(\emptyset, G_k) &= G_k[!\varepsilon] \\
\Pi(\varepsilon, G_k) &= G_k \\
\Pi(a, G_k) &= G_k[a p_k] \\
\Pi(e_1 e_2, G_k) &= \Pi(e_1, \Pi(e_2, G_k)) \\
\Pi(e_1 | e_2, G_k) &= G_2[p_1 / p_2], \text{ onde } G_2 = (V_2, T, P_2, p_2) = \Pi(e_2, (V_1, T, P_1, p_k)) \text{ e} \\
&\quad (V_1, T, P_1, p_1) = \Pi(e_1, G_k) \\
\Pi(e_1^*, G_k) &= G, \text{ onde } G = (V_1, T, P_1 \cup \{A \rightarrow p_1 / p_k\}, A), \\
&\quad (V_1, T, P_1, p_1) = \Pi(e_1, (V_k \cup \{A\}, T, P_k, A)) \text{ e} \\
&\quad A \notin V_k
\end{aligned}$$

Figura 3.2: Definição da Função Π , onde $G_k = (V_k, T, P_k, p_k)$

dessa expressão de parsing é sempre `fail`, de modo que $L(G_k[!\varepsilon]) = \emptyset$.

No caso de uma expressão regular ε , o resultado da função Π é a mesma gramática G_k que fornecemos a ela.

Quando a expressão regular é da forma a , a função Π nos dá uma gramática com o mesmo conjunto de terminais, não terminais e produções de G_k , e que possui $a p_k$ como sua expressão de parsing inicial.

Quando a expressão regular fornecida a Π é uma concatenação $e_1 e_2$, primeiro criamos uma gramática $\Pi(e_2, G_k)$, que é equivalente a $e_2 e_k$, e depois a usamos como continuação para obter uma PEG que é equivalente à expressão regular $e_1 (e_2 e_k)$.

Se a expressão regular é uma escolha $e_1 | e_2$, criamos uma gramática que possui p_1 / p_2 como sua expressão de parsing inicial, e que é equivalente à expressão regular $e_1 e_k | e_2 e_k$. Ao criarmos essa gramática, usamos uma gramática intermediária, pois a transformação de e_1 pode introduzir novos não terminais, que não devem ser usados durante a transformação de e_2 , e vice versa. Ao invés de transformarmos primeiro e_1 e então e_2 , poderíamos ter transformado primeiro e_2 e então e_1 .

Finalmente, quando a expressão regular é uma repetição e_1^* , criamos uma gramática G que é equivalente à expressão regular $e_1 e_1^* e_k | e_k$. Ao criarmos G adicionamos um novo não terminal A à gramática G_k , de modo que $A \notin V_k$. A produção associada a esse não terminal é $A \rightarrow p_1 / p_k$, onde temos que $G[p_1]$ é equivalente a $e_1 e_1^* e_k$, e que $G[p_k]$ é equivalente a e_k .

A PEG resultante da transformação Π é linear à direita. No próximo capítulo, na seção 4.4, discutiremos a correspondência entre CFGs lineares à direita e PEGs.

3.4.1

Exemplos de Uso da Transformação Π

Vamos ver agora alguns exemplos de uso da transformação Π . Na discussão a seguir, usaremos o alfabeto $\Sigma = \{a, b, c\}$, e a gramática $G_k = (\emptyset, \Sigma, \emptyset, \varepsilon)$.

No primeiro exemplo, vamos usar a expressão regular e_0 a seguir, que casa cadeias que possuem pelo menos um **a**:

$$(a \mid b \mid c)^* a (a \mid b \mid c)^*$$

O resultado da transformação $\Pi(e_0, G_k)$ é uma PEG que possui as seguintes produções, onde A é a expressão de parsing inicial:

$$A \rightarrow aA \mid bA \mid cA \mid aB \quad B \rightarrow aB \mid bB \mid cB \mid \varepsilon$$

Quando a expressão regular e_0 casa uma dada entrada, não sabemos quantos **a**'s a primeira repetição de e_0 casa, pois o casamento de uma expressão regular é não determinístico. Por outro lado, como o casamento de uma PEG é determinístico, temos que o casamento da alternativa aB do não terminal A só é bem sucedido quando o casamento da alternativa aA falha, de modo que A casa um prefixo da entrada que se estende até o último caractere **a**, e portanto o não terminal B não casa nenhum **a**.

A expressão regular a seguir define a mesma linguagem da expressão regular anterior:

$$(b \mid c)^* a (a \mid b \mid c)^*$$

Para esta expressão regular, obtemos a seguinte gramática G' como resultado da transformação Π , onde A é a expressão de parsing inicial:

$$A \rightarrow bA \mid cA \mid aB \quad B \rightarrow aB \mid bB \mid cB \mid \varepsilon$$

Embora a gramática G' seja similar à gramática G , um reconhecedor para G' baseado em backtracking apresenta um desempenho melhor do que um reconhecedor correspondente para G . A razão para isso é que o reconhecedor de G analisa mais de uma vez a porção da entrada que segue o último **a**, enquanto que o reconhecedor de G' não.

No próximo exemplo, vamos definir uma expressão regular e_0 que casa cadeias onde todo **a** é seguido por no mínimo um **b** ou um **c**. A definição de e_0 é apresentada a seguir:

$$(b \mid c)^* (a (b \mid c) (b \mid c)^*)^*$$

O resultado da transformação $\Pi(e_0, G_k)$ é uma gramática G que possui as seguintes produções, onde A é a expressão de parsing inicial:

$$A \rightarrow bA / cA / B \quad B \rightarrow a(bC / cC) / \varepsilon \quad C \rightarrow bC / cC / B$$

Como a expressão regular e_0 possui três repetições, a PEG G obtida a partir da transformação Π possui três não terminais. Dada a entrada **abaca**, o resultado do casamento de e_0 pode ser ε , **ab**, ou **abac**. Dada a mesma entrada, o resultado do casamento da gramática G é **abac**, pois o resultado do casamento de uma PEG para uma dada entrada é único.

3.5

Transformação de Repetições e_1^* onde e_1 Casa a Cadeia Vazia

Dada uma expressão regular e_0 , essa expressão pode ter subexpressões da forma e_1^* onde e_1 casa a cadeia vazia. Nesse caso, como o casamento de e_1 pode ser bem sucedido e não consumir nenhum caractere da entrada, temos que a expressão e_1^* pode casar um número infinito de vezes.

Para evitar que uma repetição infinita ocorra, bibliotecas de casamento de padrões baseadas em backtracking, como a de Perl [Wall, 2000], implementam políticas para interromper um casamento quando a expressão regular que está sendo repetida casa a cadeia vazia [perldoc].

Dada uma expressão regular e_0 , iremos dizer que e_0 é uma expressão regular bem formada se ela não possui subexpressões e_1^* onde $\varepsilon \in L(e_1)$.

Quando uma expressão regular não é bem formada, a PEG obtida através da transformação Π não é completa. Para ver um exemplo disso, vamos transformar a seguinte expressão regular e_0 , que é mal formada, em uma PEG:

$$(a \mid \varepsilon)^* b$$

A gramática G resultante da transformação $\Pi(e_0, \varepsilon)$, possui a seguinte produção, que é recursiva à esquerda:

$$A \rightarrow aA / A / b$$

Como G possui produções recursivas à esquerda, G não é uma gramática completa. Na próxima seção, veremos que quando uma expressão regular é bem formada, a gramática obtida através da transformação Π é completa.

No restante desta seção, vamos discutir como podemos reescrever uma expressão regular para torná-la bem formada, e assim evitar o casamento infinito de uma repetição e a geração de uma PEG que não é completa.

$$\begin{aligned}
isNull(\emptyset) &= \mathbf{false} \\
isNull(\varepsilon) &= \mathbf{true} \\
isNull(a) &= \mathbf{false} \\
isNull(e_1 e_2) &= isNull(e_1) \wedge isNull(e_2) \\
isNull(e_1 | e_2) &= isNull(e_1) \wedge isNull(e_2) \\
isNull(e_1^*) &= isNull(e_1)
\end{aligned}$$

Figura 3.3: Definição da Função $isNull$

Uma abordagem que poderíamos adotar para obter uma expressão regular bem formada seria definir uma função f que reescreve uma expressão regular de maneira *bottom-up* com base nas seguintes igualdades, onde e_1 não casa a cadeia vazia:

$$L(\varepsilon^*) = L(\varepsilon) \quad L((e_1 | \varepsilon)^*) = L(e_1^*) \quad L((e_1^*)^*) = L(e_1^*)$$

Nessa abordagem, restringiríamos as possíveis formas de uma expressão regular, de modo que o resultado de f seria uma expressão em que todas as subexpressões teriam uma das seguintes formas, onde e_1 não casa a cadeia vazia:

$$\varepsilon \quad e_1 | \varepsilon \quad e_1^* \quad e_1$$

Dado que uma expressão regular poderia estar em uma das quatro formas acima, a função f teria 16 casos para tratar da concatenação, e mais 16 casos para tratar da escolha. Além do grande número de casos necessário para definir a função f , um outro ponto negativo dessa abordagem é que muitas vezes a função f nos daria uma expressão regular bem formada maior do que a expressão regular original.

Como a definição da função f teria muitos casos e o seu resultado às vezes seria uma expressão regular maior do que a original, vamos seguir uma outra abordagem para transformar uma expressão regular e_0 em uma expressão bem formada. Nesta abordagem, vamos usar as funções auxiliares $isNull$, cuja definição aparece na figura 3.3, e $hasEmpty$, cuja definição aparece na figura 3.4.

A função $isNull$ nos diz se a linguagem associada a uma expressão regular contém somente a cadeia vazia. Por sua vez, a função $hasEmpty$ nos diz se a linguagem associada a uma expressão regular contém a cadeia vazia.

Com a ajuda das funções $isNull$ e $hasEmpty$, iremos definir as funções f_{out} e f_{in} , que reescrevem uma expressão regular de maneira *top-down*. O uso

$$\begin{aligned}
hasEmpty(\emptyset) &= \mathbf{false} \\
hasEmpty(\varepsilon) &= \mathbf{true} \\
hasEmpty(a) &= \mathbf{false} \\
hasEmpty(e_1 e_2) &= hasEmpty(e_1) \wedge hasEmpty(e_2) \\
hasEmpty(e_1 | e_2) &= hasEmpty(e_1) \vee hasEmpty(e_2) \\
hasEmpty(e_1^*) &= \mathbf{true}
\end{aligned}$$

Figura 3.4: Definição da Função *hasEmpty*

$$\begin{aligned}
f_{out}(\emptyset) &= \emptyset \\
f_{out}(\varepsilon) &= \varepsilon \\
f_{out}(a) &= a \\
f_{out}(e_1 e_2) &= f_{out}(e_1) f_{out}(e_2) \\
f_{out}(e_1 | e_2) &= f_{out}(e_1) | f_{out}(e_2) \\
f_{out}(e_1^*) &= \begin{cases} f_{out}(e_1)^* & \text{se } \neg hasEmpty(e_1) \\ \varepsilon & \text{se } isNull(e_1) \\ f_{in}(e_1)^* & \text{caso contrário} \end{cases}
\end{aligned}$$

Figura 3.5: Definição da Função *f_{out}*

de uma abordagem top-down resultará em uma expressão regular bem formada mais simples do que a expressão original.

Dada uma expressão regular e_0 , vamos usar a função f_{out} , cuja definição é apresentada na figura 3.5, para encontrar subexpressões e_1^* onde e_1 casa a cadeia vazia. Quando encontra uma dessas subexpressões, f_{out} usa a função f_{in} , definida na figura 3.6, para reescrever a expressão e_1 de modo que $f_{in}(e_1)$ não case a cadeia vazia e que $f_{in}(e_1)^*$ defina a mesma linguagem que e_1^* .

Com base na definição de f_{out} , dada uma expressão regular e_0 , temos que:

- $L(e_0) = L(f_{out}(e_0))$
- $f_{out}(e_0)$ é bem formada

Já com base na definição de f_{in} , dada uma expressão regular e_0 que casa a cadeia vazia, onde $L(e_0) \neq \{\varepsilon\}$, temos que:

- $\varepsilon \notin L(f_{in})$
- $L(e_0^*) = L(f_{in}(e_0)^*)$
- $f_{in}(e_0)$ é bem formada

$$\begin{aligned}
f_{in}(e_1 e_2) &= f_{in}(e_1 | e_2) \\
f_{in}(e_1 | e_2) &= \begin{cases} f_{in}(e_2) & \text{se } isNull(e_1) \text{ e } hasEmpty(e_2) \\ f_{out}(e_2) & \text{se } isNull(e_1) \text{ e } \neg hasEmpty(e_2) \\ f_{in}(e_1) & \text{se } hasEmpty(e_1) \text{ e } isNull(e_2) \\ f_{out}(e_1) & \text{se } \neg hasEmpty(e_1) \text{ e } isNull(e_2) \\ f_{out}(e_1) | f_{in}(e_2) & \text{se } \neg hasEmpty(e_1) \text{ e } \neg isNull(e_2) \\ f_{in}(e_1) | f_{out}(e_2) & \text{se } \neg isNull(e_1) \text{ e } \neg hasEmpty(e_2) \\ f_{in}(e_1) | f_{in}(e_2) & \text{caso contrário} \end{cases} \\
f_{in}(e_1^*) &= \begin{cases} f_{in}(e_1) & \text{se } hasEmpty(e_1) \\ f_{out}(e_1) & \text{caso contrário} \end{cases}
\end{aligned}$$

Figura 3.6: Definição da Função $f_{in}(e_0)$, onde $\neg isNull(e_0)$ e $hasEmpty(e_0)$

É fácil mostrar que as afirmações anteriores sobre f_{out} e f_{in} são verdadeiras. A parte não óbvia da prova é quando a função f_{in} recebe uma expressão da forma $e_1 e_2$.

Nesse caso, podemos ver na definição de f_{in} que o resultado de $f_{in}(e_1 e_2)$ é dado por $f_{in}(e_1 | e_2)$. Essa definição usa o fato de que a expressão regular recebida por f_{in} sempre casa a cadeia vazia. Desse modo, temos que $\varepsilon \in e_1 e_2$, e portanto sabemos que $\varepsilon \in (e_1)$ e que $\varepsilon \in (e_2)$. Assim, a seguinte igualdade é verdadeira:

$$L((e_1 e_2)^*) = L((e_1 | e_2)^*) , \text{ onde } \varepsilon \in L(e_1 e_2)$$

Como um exemplo, vamos usar as funções f_{out} e f_{in} para transformar a expressão regular $((a | \varepsilon) b^*)^*$ em uma expressão bem formada. A seguir, mostramos a sequência de passos dessa transformação:

$$\begin{aligned}
f_{out}(((a | \varepsilon) b^*)^*) &= f_{in}((a | \varepsilon) b^*)^* \\
&= f_{in}((a | \varepsilon) | b^*)^* \\
&= (f_{in}(a | \varepsilon) | f_{in}(b^*))^* \\
&= (f_{out}(a) | f_{out}(b))^* \\
&= (a | b)^*
\end{aligned}$$

Como podemos ver, as funções f_{out} e f_{in} nos deram expressões bem formadas que estão de acordo com as condições anteriores.

3.6

Corretude da Transformação Π

Nesta seção iremos provar que dada uma expressão regular e_0 e uma PEG $G_k = (V_k, T, P_k, p_k)$, que é equivalente a uma expressão regular e_k , a transformação $\Pi(e_0, G_k)$ nos dá uma PEG que é equivalente a $e_0 e_k$. Primeiro iremos definir um lema auxiliar, e depois vamos mostrar que dada uma cadeia x temos que $\Pi(e_0, G_k) \stackrel{\leq x}{\Rightarrow} e_0 e_k$ e também que $\Pi(e_0, G_k) \stackrel{\leq x}{\Leftarrow} e_0 e_k$. Em seguida, vamos apresentar uma proposição sobre a equivalência entre expressões regulares e PEGs. Por fim, iremos discutir quando uma expressão regular e uma PEG equivalente definem a mesma linguagem.

Nas provas a seguir, usaremos indução na complexidade de um par (e_0, x) , onde e_0 é uma expressão regular e x é uma cadeia de terminais. A complexidade de um par (e_0, x) é dada pela estrutura de e_0 e pelo comprimento de x . Assim, dados os pares (e_1, x_1) e (e_2, x_2) , temos que a complexidade do primeiro par é maior do que a do segundo se a estrutura de e_1 é maior do que a estrutura de e_2 , ou se a estrutura de e_1 é igual à estrutura de e_2 e $|x_1| > |x_2|$.

Nas provas a seguir, usaremos o fato de que todas as produções da gramática G_k estão presentes na gramática $\Pi(e_0, G_k)$, independentemente da expressão regular e_0 .

Vamos começar definindo o seguinte lema, que nos diz que se $\Pi(e_0, G_k)$ casa uma cadeia, então G_k casa um sufixo dessa cadeia:

Lema 3.6.1. *Dada uma expressão regular e_0 bem formada, uma gramática G_k , e uma cadeia x , onde $\Pi(e_0, G_k) \stackrel{\text{PEG}}{\rightsquigarrow} y$, então existe $x' \preceq x$ tal que $G_k \stackrel{\text{PEG}}{\rightsquigarrow} y$.*

Demonstração. A prova é por indução na complexidade do par (e_0, x) .

Quando $e_0 = \varepsilon$, o resultado de $\Pi(\varepsilon, G_k)$ é a própria gramática G_k . Assim, temos que $G_k \stackrel{\text{PEG}}{\rightsquigarrow} y$, onde $x = x'$.

Quando $e_0 = a$, o resultado de $\Pi(a, G_k)$ é a gramática $G_k[a p_k]$. Desse modo, dada uma cadeia $x = ax'$, quando $G_k[a p_k] \stackrel{\text{PEG}}{\rightsquigarrow} y$, a regra *con.1* nos diz que $G_k[a] \stackrel{\text{PEG}}{\rightsquigarrow} x'$ e que $G_k[p_k] \stackrel{\text{PEG}}{\rightsquigarrow} y$.

Quando a expressão regular e_0 é da forma $e_1 e_2$, temos a transformação $\Pi(e_1 e_2, G_k) = \Pi(e_1, \Pi(e_2, G_k))$. Pela hipótese de indução temos que $\Pi(e_2, G_k) \stackrel{\text{PEG}}{\rightsquigarrow} y$, e novamente pela hipótese de indução concluímos que $G_k \stackrel{\text{PEG}}{\rightsquigarrow} x''$, onde x'' é um sufixo de x' .

Quando $e_0 = e_1 | e_2$, a transformação Π nos dá uma gramática G_2 cuja expressão de parsing inicial é p_1 / p_2 . Há duas regras na semântica de $\stackrel{\text{PEG}}{\rightsquigarrow}$ que podem ter sido usadas no casamento de p_1 / p_2 : *ord.1* e *ord.2*.

Se a regra *ord.1* foi usada, então o casamento de $G_2[p_1]$ foi bem sucedido, e portanto sabemos que o casamento de p_1 também é bem sucedido na gramática $(V_1, T, P_1, p_1) = \Pi(e_1, G_k)$. Assim, pela hipótese de indução, concluímos que $G_k x' \overset{\text{PEG}}{\rightsquigarrow} y$. A prova é similar se a regra *ord.2* foi usada.

Finalmente, quando temos uma repetição e_1^* , o resultado de $\Pi(e_1^*, G_k)$ é uma gramática $G = (V_1, T, P, A)$, onde $A \rightarrow p_1/p_k$. Pela regra *var.1* sabemos que se $G[A] x \overset{\text{PEG}}{\rightsquigarrow} y$ então temos que $G[p_1/p_k] x \overset{\text{PEG}}{\rightsquigarrow} y$. Há duas regras em $\overset{\text{PEG}}{\rightsquigarrow}$ que podem ter sido usadas no casamento dessa escolha: *ord.1* e *ord.2*.

Se a regra *ord.2* foi usada, então $G[p_k] x' \overset{\text{PEG}}{\rightsquigarrow} y$, onde $x = x'$. Assim, temos que o casamento de p_k também é bem sucedido em G_k , e podemos concluir que $G_k[p_k] x' \overset{\text{PEG}}{\rightsquigarrow} y$.

Se a regra *ord.1* foi usada, então $\Pi(e_1, \Pi(e_1^*, G_k)) x \overset{\text{PEG}}{\rightsquigarrow} y$, e pela hipótese de indução temos que $\Pi(e_1^*, G_k) x' \overset{\text{PEG}}{\rightsquigarrow} y$. Nesse caso, não podemos fazer indução na estrutura de e_1^* , então vamos fazer indução no comprimento de x . Dado que e_1 casa um prefixo não vazio da entrada, temos que $x' \prec x$, e pela hipótese de indução concluímos que $G_k x'' \overset{\text{PEG}}{\rightsquigarrow} y$, onde x'' é um sufixo de x' . \square

O próximo lema nos diz quando uma expressão regular da forma $e_0 e_k$ casa como a gramática $\Pi(e_0, G_k)$:

Lema 3.6.2. *Seja x uma cadeia de caracteres, e_k uma expressão regular, e G_k uma gramática, onde $G_k \overset{\prec x}{\Rightarrow} e_k$. Então para qualquer expressão e_0 bem formada temos que $\Pi(e_0, G_k) \overset{\prec x}{\Rightarrow} e_0 e_k$.*

Demonstração. A prova é por indução na complexidade do par (e_0, x) .

Quando $e_0 = \varepsilon$, temos a transformação $\Pi(\varepsilon, G_k) = G_k$. Dado que $\varepsilon e_k \equiv e_k$, concluímos que $G_k \overset{\prec x}{\Rightarrow} \varepsilon e_k$.

Quando $e_0 = a$, a transformação correspondente é $\Pi(a, G_k) = G_k[a p_k]$. Dado que $G_k[a] \overset{\prec x}{\Rightarrow} a$ e que $G_k[p_k] \overset{\prec x}{\Rightarrow} e_k$, pela regra *con.1* concluímos que $G_k[a p_k] \overset{\prec x}{\Rightarrow} a e_k$.

Quando $e_0 = e_1 e_2$, a transformação correspondente é $\Pi(e_1 e_2, G_k) = \Pi(e_1, \Pi(e_2, G_k))$. Pela hipótese de indução $\Pi(e_2, G_k) \overset{\prec x}{\Rightarrow} e_2 e_k$, e novamente pela hipótese de indução temos que $\Pi(e_1, \Pi(e_2, G_k)) \overset{\prec x}{\Rightarrow} e_1 (e_2 e_k)$. Dado que $(e_1 e_2) e_k \equiv e_1 (e_2 e_k)$, podemos concluir que $\Pi(e_1 e_2, G_k) \overset{\prec x}{\Rightarrow} (e_1 e_2) e_k$.

Quando $e_0 = e_1 | e_2$, a transformação Π nos dá uma gramática G_2 cuja expressão de parsing inicial é p_1/p_2 . Há duas regras que podem ter sido usadas no casamento desta escolha: *ord.1* e *ord.2*.

Se a regra *ord.1* foi usada, então o casamento de $G_2[p_1]$ foi bem sucedido, e portanto o casamento de p_1 também é bem sucedido na gramática $(V_1, T, P_1, p_1) = \Pi(e_1, G_k)$. Pela hipótese de indução temos que $\Pi(e_1, G_k) \overset{\prec x}{\Rightarrow} e_1 e_k$, e pela regra *choice.1* temos que $\Pi(e_1 | e_2, G_k) \overset{\prec x}{\Rightarrow} e_1 e_k | e_2 e_k$. Dado que

$e_1 e_k \mid e_2 e_k \equiv (e_1 \mid e_2) e_k$, podemos concluir que $\Pi(e_1 \mid e_2, G_k) \stackrel{\leq x}{\rightarrow} (e_1 \mid e_2) e_k$. A prova é similar se a regra *ord.2* foi usada.

Finalmente, quando temos uma repetição e_1^* , a transformação $\Pi(e_1^*, G_k)$ nos dá uma gramática $G = (V_1, T, P, A)$, onde $A \rightarrow p_1 / p_k$. Pela regra *var.1* sabemos que se $G[A] \ x \stackrel{\text{PEG}}{\rightsquigarrow} y$, então temos que $G[p_1 / p_k] \ x \stackrel{\text{PEG}}{\rightsquigarrow} y$. Há duas regras que podem ter sido usadas no casamento da escolha anterior: *ord.1* e *ord.2*.

Se a regra *ord.2* foi usada, então o casamento de $G[p_k]$ foi bem sucedido, e portanto o casamento de $G_k[p_k]$ também é bem sucedido. Como $G_k[p_k] \stackrel{\leq x}{\rightarrow} e_k$, pela regra *choice.2* temos que $\Pi(e_1^*, G_k) \stackrel{\leq x}{\rightarrow} e_1 e_1^* e_k \mid e_k$. Dado que $e_1 e_1^* e_k \mid e_k \equiv e_1^* e_k$, podemos concluir que $\Pi(e_1^*, G_k) \stackrel{\leq x}{\rightarrow} e_1^* e_k$.

Se a regra *ord.1* foi usada, temos que $\Pi(e_1, \Pi(e_1^*, G_k)) \ x \stackrel{\text{PEG}}{\rightsquigarrow} y$, e pelo lema 3.6.1 sabemos que $\Pi(e_1^*, G_k) \ x' \stackrel{\text{PEG}}{\rightsquigarrow} y$. Dado que e_1 casa um prefixo não vazio da entrada, então $x' \prec x$, e podemos fazer indução no comprimento da cadeia de entrada. Assim, pela hipótese de indução temos que $\Pi(e_1^*, G_k) \stackrel{\leq x}{\rightarrow} e_1^* e_k$, e novamente pela hipótese de indução temos que $\Pi(e_1, \Pi(e_1^*, G_k)) \stackrel{\leq x}{\rightarrow} e_1 (e_1^* e_k)$. Portanto, pela regra *choice.1* temos que $\Pi(e_1^*, G_k) \stackrel{\leq x}{\rightarrow} e_1 e_1^* e_k \mid e_k$. Dado que $e_1 e_1^* e_k \mid e_k \equiv e_1^* e_k$, podemos concluir que $\Pi(e_1^*, G_k) \stackrel{\leq x}{\rightarrow} e_1^* e_k$. \square

Agora vamos provar um lema que complementa o lema anterior. O lema a seguir nos diz que podemos usar a função Π para obter uma PEG que casa quando uma dada expressão regular casa:

Lema 3.6.3. *Seja x uma cadeia de caracteres, e_k uma expressão regular, e G_k uma gramática, onde $G_k \stackrel{\leq x}{\leftarrow} e_k$. Então para qualquer expressão e_0 bem formada temos que $\Pi(e_0, G_k) \stackrel{\leq x}{\leftarrow} e_0 e_k$.*

Demonstração. A prova é por indução na complexidade do par (e_0, x) .

Quando $e_0 = \varepsilon$, temos que $e_0 e_k = \varepsilon e_k \equiv e_k$. Dado que $\Pi(\varepsilon, G_k) = G_k$, concluimos que $\Pi(\varepsilon, G_k) \stackrel{\leq x}{\leftarrow} \varepsilon e_k$.

Quando $e_0 = a$, então $e_0 e_k = a e_k$. Dado que $G_k[a] \stackrel{\leq x}{\leftarrow} a$ e que $G_k[p_k] \stackrel{\leq x}{\leftarrow} e_k$, pela regra *con.1* temos que $G_k[a p_k] \stackrel{\leq x}{\leftarrow} a e_k$. Como $\Pi(a, G_k) = G_k[a p_k]$, podemos concluir que $\Pi(a, G_k) \stackrel{\leq x}{\leftarrow} a e_k$.

Quando $e_0 = e_1 e_2$, temos que $e_0 e_k = (e_1 e_2) e_k \equiv e_1 (e_2 e_k)$. Pela hipótese de indução temos que $\Pi(e_2, G_k) \stackrel{\leq x}{\leftarrow} e_2 e_k$, e novamente pela hipótese de indução temos que $\Pi(e_1, \Pi(e_2, G_k)) \stackrel{\leq x}{\leftarrow} e_1 (e_2 e_k)$. Dado que $\Pi(e_1 e_2, G_k) = \Pi(e_1, \Pi(e_2, G_k))$, podemos concluir que $\Pi(e_1 e_2, G_k) \stackrel{\leq x}{\leftarrow} (e_1 e_2) e_k$.

Quando $e_0 = e_1 \mid e_2$, temos que $e_0 e_k = (e_1 \mid e_2) e_k \equiv e_1 e_k \mid e_2 e_k$. Há duas regras associadas ao casamento da escolha $e_1 \mid e_2$: *choice.1* e *choice.2*.

Se a regra *choice.1* foi usada, então o casamento de e_1 foi bem sucedido. Dado que $(V_1, T, P_1, p_1) = \Pi(e_1, G_k)$, pela hipótese de indução temos que

$\Pi(e_1, G_k)[p_1] \stackrel{\prec x}{\Leftarrow} e_1 e_k$. Como o casamento de p_1 também é bem sucedido na gramática $(V_2, T, P_2, p_1 / p_2) = \Pi(e_1 | e_2, G_k)$, pela regra *ord.1* concluímos que $\Pi(e_1 | e_2, G_k) \stackrel{\prec x}{\Leftarrow} (e_1 | e_2) e_k$.

Se a regra *choice.2* foi usada, então o casamento de e_2 foi bem sucedido. Dado que o casamento de e_2 foi bem sucedido, vamos analisar dois casos: quando o casamento de e_1 é bem sucedido, e quando o casamento de e_1 não é bem sucedido.

Se o casamento de e_1 é bem sucedido, a prova é similar a quando a regra *choice.1* foi usada.

Se o casamento de e_1 não é bem sucedido, como a gramática $(V_1, T, P_1, p_1) = \Pi(e_1, G_k)$ é completa, pela contrapositiva do lema 3.6.2 temos que o casamento de $\Pi(e_1, G_k)[p_1]$ não é bem sucedido. Dado que $(V_2, T, P_2, p_2) = \Pi(e_2, \Pi(e_1, G_k)[p_k])$, sabemos que o casamento de p_1 também não é bem sucedido nessa gramática, e como o casamento de e_2 é bem sucedido, pela hipótese de indução temos que $\Pi(e_2, \Pi(e_1, G_k)[p_k]) \stackrel{\prec x}{\Leftarrow} e_2 e_k$. Assim, pela regra *ord.2*, concluímos que $\Pi(e_1 | e_2, G_k) \stackrel{\prec x}{\Leftarrow} (e_1 | e_2) e_k$.

Finalmente, no caso de uma repetição, temos a expressão regular $e_1^* e_k \equiv e_1 e_1^* e_k | e_k$. Há duas regras que podem ter sido usadas no casamento dessa escolha: *choice.1* e *choice.2*.

Se a regra *choice.1* foi usada, pela regra *con.1* sabemos que o casamento de e_1 é bem sucedido. Dado que e_1 casa uma cadeia não vazia, temos que o casamento de e_1^* é bem sucedido para um sufixo $x' \prec x$. Assim, pela hipótese de indução, temos que $\Pi(e_1^*, G_k) \stackrel{\prec x}{\Leftarrow} e_1^* e_k$. Novamente pela hipótese de indução, temos que $\Pi(e_1, \Pi(e_1^*, G_k)) \stackrel{\prec x}{\Leftarrow} e_1 e_1^* e_k$. Seja p_1 a expressão de parsing inicial de $\Pi(e_1, \Pi(e_1^*, G_k))$, pela regra *ord.1* temos que $\Pi(e_1^*, G_k)[p_1 / p_k] \stackrel{\prec x}{\Leftarrow} e_1 e_1^* e_k | e_k$, e pela regra *var.1* concluímos que $\Pi(e_1^*, G_k) \stackrel{\prec x}{\Leftarrow} e_1^* e_k$.

Se a regra *choice.2* foi usada, precisamos analisar se o casamento da primeira alternativa da escolha é bem sucedido ou não.

Quando o casamento da primeira alternativa da escolha é bem sucedido, então a prova é análoga a quando a regra *choice.1* foi usada. Vamos provar então o caso em que o casamento da primeira alternativa da escolha não é bem sucedido.

Seja p_1 a expressão de parsing inicial de $\Pi(e_1, \Pi(e_1^*, G_k))$, como a gramática $\Pi(e_1^*, G_k)$ é completa, pela contrapositiva do lema 3.6.2 temos que o casamento de $\Pi(e_1^*, G_k)[p_1]$ não é bem sucedido. Dado que $G_k[p_k] \stackrel{\prec x}{\Leftarrow} e_k$, então temos que $\Pi(e_1^*, G_k)[p_k] \stackrel{\prec x}{\Leftarrow} e_k$. Assim, pela regra *ord.2*, temos que $\Pi(e_1^*, G_k)[p_1 / p_k] \stackrel{\prec x}{\Leftarrow} e_1 e_1^* e_k | e_k$, e pela regra *var.1* concluímos que $\Pi(e_1^*, G_k) \stackrel{\prec x}{\Leftarrow} e_1^* e_k$. \square

Agora, vamos usar os lemas 3.6.2 e 3.6.3 para provar a seguinte proposição a respeito da equivalência entre uma dada expressão regular e_0 e a PEG obtida a partir de e_0 usando a transformação Π .

Proposição 3.6.4. *Dada uma expressão regular e_0 bem formada, temos que a expressão regular $e_0 \varepsilon \equiv e_0$ é equivalente à gramática $\Pi(e_0, \varepsilon)$.*

Demonstração. Como a expressão regular ε é equivalente à expressão de parsing ε , pelo lema 3.6.2 para qualquer cadeia x temos que $\Pi(e_0, \varepsilon) \xrightarrow{\varepsilon} e_0 \varepsilon$, e pelo lema 3.6.3 para qualquer cadeia x temos que $\Pi(e_0, \varepsilon) \xrightarrow{x} e_0 \varepsilon$. \square

Como um corolário da proposição acima, dado que $a \varepsilon \equiv a$ e que $\Pi(a, \varepsilon) = a$, seja e_0 uma expressão regular bem formada então $e_0 a$ é equivalente à gramática $\Pi(e_0, a)$.

Apesar de termos estabelecido a equivalência entre expressões regulares e PEGs, ainda não discutimos quando uma expressão regular e uma PEG definem a mesma linguagem.

Anteriormente, na proposição 3.2.2, mostramos que $x \in L(e_0)$ se e somente se $e_0 xy \xrightarrow{\text{RE}} y$. Nessa definição, o casamento da expressão e_0 usando $\xrightarrow{\text{RE}}$ não precisa casar toda a entrada. Contudo, tal definição não nos permite mostrar que uma expressão regular e uma PEG equivalente definem a mesma linguagem, pois uma expressão regular pode se relacionar com diferentes sufixos da entrada através da relação $\xrightarrow{\text{RE}}$, enquanto que uma PEG equivalente irá se relacionar com apenas um sufixo da entrada.

Podemos ver isso no exemplo a seguir. Dada a expressão regular $a|ab$ e a PEG equivalente a/ab , para qualquer cadeia da forma ax temos a seguinte árvore de prova associada ao casamento da PEG:

$$\frac{\frac{\text{---}(\text{char.1})}{G[a] \quad ax \xrightarrow{\text{PEG}} x}}{G[a/b] \quad ax \xrightarrow{\text{PEG}} x}(\text{ord.1})$$

Desse modo, a cadeia $ab \notin L(a/ab)$, porém $ab \in L(a|ab)$. Assim, precisamos mudar as definições de linguagens de expressões regulares e de PEGs para poder estabelecer a correspondência entre a linguagem de uma expressão regular e de uma PEG equivalente.

Uma forma de fazer isso é usar o símbolo $\$ \notin T$ como um marcador de final da entrada, de modo que uma expressão regular e_0 se relacione com apenas um sufixo de uma determinada entrada. Dessa forma, dada uma expressão regular e_0 e uma entrada x , se $e_0 \$ x \$ \xrightarrow{\text{RE}} \varepsilon$ então $x \in L(e_0)$.

No caso de PEGs, dada uma entrada da forma $x\$$ só há um casamento bem sucedido de uma PEG $\Pi(e_0, \$)$ quando toda a entrada é consumida.

Assim, o casamento de uma alternativa de uma escolha da PEG $\Pi(e_0, \$)$ é bem sucedido somente se toda a entrada é consumida, como a árvore de prova a seguir mostra (omitimos as regras usadas por motivo de espaço):

$$\begin{array}{c}
 \frac{G[a] \text{ ab\$ } \overset{\text{PEG}}{\rightsquigarrow} \text{ b\$} \quad \frac{G[\$] \text{ b\$ } \overset{\text{PEG}}{\rightsquigarrow} \text{ fail}}{G[a \$] \text{ ab\$ } \overset{\text{PEG}}{\rightsquigarrow} \text{ fail}} \quad \frac{G[a] \text{ ab\$ } \overset{\text{PEG}}{\rightsquigarrow} \text{ b\$}}{G[a \$ / a b \$] \text{ ab\$ } \overset{\text{PEG}}{\rightsquigarrow} \varepsilon} \quad \frac{\frac{G[b\$] \text{ b\$ } \overset{\text{PEG}}{\rightsquigarrow} \$} \quad \frac{G[\$] \$ \overset{\text{PEG}}{\rightsquigarrow} \varepsilon}}{G[b \$] \text{ b\$ } \overset{\text{PEG}}{\rightsquigarrow} \varepsilon}}{G[a b \$] \text{ ab\$ } \overset{\text{PEG}}{\rightsquigarrow} \varepsilon} \\
 \hline
 G[a \$ / a b \$] \text{ ab\$ } \overset{\text{PEG}}{\rightsquigarrow} \varepsilon
 \end{array}$$

Com base nisso, vamos definir que se $\Pi(e_0, \$) \ x \$ \overset{\text{PEG}}{\rightsquigarrow} \varepsilon$ então $x \in L(\Pi(e_0, \$))$.

Portanto, seja e_0 uma expressão regular bem formada, temos que $e_0 \$ \ x \$ \overset{\text{RE}}{\rightsquigarrow} \varepsilon$ se e somente se $G[\Pi(e_0, \$)] \ x \$ \overset{\text{PEG}}{\rightsquigarrow} \varepsilon$. Logo, a expressão regular e_0 e a PEG $\Pi(e_0, \$)$ definem a mesma linguagem.

Existe uma outra forma de estabelecer a correspondência entre a linguagem de uma expressão regular e de uma PEG. Nessa forma, dada uma expressão regular e_0 e uma entrada x , definimos que se $e_0 \ x \overset{\text{RE}}{\rightsquigarrow} \varepsilon$ então $x \in L(e_0)$. Para definir a linguagem de uma PEG, vamos usar a expressão de parsing $!$ que discutimos na seção 2.1 e que é um açúcar sintático para casar qualquer terminal da entrada. Como o casamento da expressão de parsing $!$ é bem sucedido somente quando a entrada é vazia, podemos usar essa expressão para testar se a PEG casou toda a entrada. Assim, vamos definir que se $\Pi(e_0, !.) \ x \overset{\text{PEG}}{\rightsquigarrow} \varepsilon$ então $x \in \Pi(e_0, \$)$.

Portanto, dada uma expressão regular e_0 bem formada e uma cadeia x , temos que $e_0 \ x \overset{\text{RE}}{\rightsquigarrow} \varepsilon$ se e somente se $\Pi(e_0, !.) \ x \overset{\text{PEG}}{\rightsquigarrow} \varepsilon$.