RE from a RG

RG with Expressions (RGE)

A regular grammar with expressions, RGE, is (V, Σ, S, P), as before, but we extend the definition to include these two types of productions, $A \to \Lambda$ or $A \to rB$, where r is a regular expression over Σ. The latter is interpreted to mean that the variable A can be replaced, in a derivation, by the string xB for any $x \in \mathcal{L}(r)$. Note r can be just Λ, so we can have $A \to \Lambda B$ (which will not be written $A \to B$, in order to emphasize the presence of an RE). The definition of the language generated by an RGE is unchanged from regular grammars.

RE from a RG

Algorithm 8.1

Constructing an RE from a regular grammar
Input: A regular grammar $G = (V, \Sigma, S, P)$.
Output: A regular expression r over Σ, such that $\mathcal{L}(r) = \mathcal{L}(G)$.

Let V' be $V \cup \{S', H\}$, where S' is the new start variable
Add $S' \to \Lambda S$ and $H \to \Lambda$ to P
for each $A \to \Lambda \in P$ **do**
  Replace $A \to \Lambda$ by $A \to \Lambda H$ in P
[Now $G' = (V', \Sigma, S', P)$ is the first RGE and $\mathcal{L}(G') = \mathcal{L}(G)$.]
for each pair from P: $D \to r_1 E, D \to r_2 E$ **do**
  Replace the pair by $D \to (r_1 + r_2)E$ in P
while $V \neq \emptyset$ **do**
  Remove some B from V
  if no $B \to rB$ in P **then**
    Add $B \to \Lambda B$ to P
  for each triple from P: $A \to r_1 B, B \to r_2 B, B \to r_3 C$ **do**
    Add $A \to (r_1(r_2)^* r_3)C$ to P
  for each pair from P: $D \to r_1 E, D \to r_2 E$ **do**
    Replace the pair by $D \to (r_1 + r_2)E$ in P
  Remove all productions using B from P
The only remaining productions are $S' \to rH$ and $H \to \Lambda$
return (r)

RE from a RG

Algorithm 8.1

Constructing an RE from a regular grammar
Input: A regular grammar $G = (V, \Sigma, S, P)$.
Output: A regular expression r over Σ, such that $\mathcal{L}(r) = \mathcal{L}(G)$.

Let V' be $V \cup \{S', H\}$, where S' is the new start variable
Add $S' \to \Lambda S$ and $H \to \Lambda$ to P
for each $A \to \Lambda \in P$ **do**
    Replace $A \to \Lambda$ by $A \to \Lambda H$ in P
[Now $G' = (V', \Sigma, S', P)$ is the first RGE and $\mathcal{L}(G') = \mathcal{L}(G)$.]
for each pair from P: $D \to r_1 E, D \to r_2 E$ **do**
    Replace the pair by $D \to (r_1 + r_2)E$ in P
while $V \neq \emptyset$ **do**
    Remove some B from V
    if no $B \to rB$ in P **then**
        Add $B \to \Lambda B$ to P
    for each triple from P: $A \to r_1 B, B \to r_2 B, B \to r_3 C$ **do**
        Add $A \to (r_1 (r_2)^* r_3)C$ to P
    for each pair from P: $D \to r_1 E, D \to r_2 E$ **do**
        Replace the pair by $D \to (r_1 + r_2)E$ in P
    Remove all productions using B from P
The only remaining productions are $S' \to rH$ and $H \to \Lambda$
return (r)

Example:
$$P = \{S \to aA,\ A \to aB,\ A \to aA,\ A \to bB,\ B \to aS,\ B \to bB,\ B \to \Lambda\}$$

RE from a RG

Algorithm 8.1

Constructing an RE from a regular grammar
Input: A regular grammar $G = (V, \Sigma, S, P)$.
Output: A regular expression r over Σ, such that $\mathcal{L}(r) = \mathcal{L}(G)$.

Let V' be $V \cup \{S', H\}$, where S' is the new start variable
Add $S' \to \Lambda S$ and $H \to \Lambda$ to P
for each $A \to \Lambda \in P$ **do**
    Replace $A \to \Lambda$ by $A \to \Lambda H$ in P
[Now $G' = (V', \Sigma, S', P)$ is the first RGE and $\mathcal{L}(G') = \mathcal{L}(G)$.]
for each pair from P: $D \to r_1 E, D \to r_2 E$ **do**
    Replace the pair by $D \to (r_1 + r_2)E$ in P
while $V \neq \emptyset$ **do**
    Remove some B from V
    if no $B \to rB$ in P **then**
        Add $B \to \Lambda B$ to P
    for each triple from P: $A \to r_1 B, B \to r_2 B, B \to r_3 C$ **do**
        Add $A \to (r_1 (r_2)^* r_3)C$ to P
    for each pair from P: $D \to r_1 E, D \to r_2 E$ **do**
        Replace the pair by $D \to (r_1 + r_2)E$ in P
    Remove all productions using B from P
The only remaining productions are $S' \to rH$ and $H \to \Lambda$
return (r)

Example:

$$P = \{S \to aA,\ A \to aB,\ A \to aA,\ A \to bB,\ B \to aS,\ B \to bB,\ B \to \Lambda\}$$

$$P' = \{S \to aA,\ A \to aB,\ A \to aA,\ A \to bB,\ B \to aS,\ B \to bB,\ B \to \Lambda H,\ S' \to \Lambda S,\ H \to \Lambda\}$$

RE from a RG

Algorithm 8.1

Constructing an RE from a regular grammar
Input: A regular grammar $G = (V, \Sigma, S, P)$.
Output: A regular expression r over Σ, such that $\mathcal{L}(r) = \mathcal{L}(G)$.

Let V' be $V \cup \{S', H\}$, where S' is the new start variable
Add $S' \to \Lambda S$ and $H \to \Lambda$ to P
for each $A \to \Lambda \in P$ **do**
    Replace $A \to \Lambda$ by $A \to \Lambda H$ in P
[Now $G' = (V', \Sigma, S', P)$ is the first RGE and $\mathcal{L}(G') = \mathcal{L}(G)$.]
for each pair from P: $D \to r_1E, D \to r_2E$ **do**
    Replace the pair by $D \to (r_1 + r_2)E$ in P
while $V \neq \emptyset$ **do**
    Remove some B from V
    if no $B \to rB$ in P **then**
        Add $B \to \Lambda B$ to P
    for each triple from P: $A \to r_1B, B \to r_2B, B \to r_3C$ **do**
        Add $A \to (r_1(r_2)^*r_3)C$ to P
    for each pair from P: $D \to r_1E, D \to r_2E$ **do**
        Replace the pair by $D \to (r_1 + r_2)E$ in P
    Remove all productions using B from P
The only remaining productions are $S' \to rH$ and $H \to \Lambda$
return (r)

Example:
$P = \{S \to aA, A \to aB, A \to aA$
$A \to bB, B \to aS, B \to bB,$
$B \to \Lambda\}$

$P' = \{S \to aA, A \to aB, A \to aA$
$A \to bB, B \to aS, B \to bB,$
$B \to \Lambda H, S' \to \Lambda S, H \to \Lambda\}$

After 2nd For Each:
$P' = \{S \to aA, A \to (a + b)B,$
$A \to aA, B \to aS, B \to bB,$
$B \to \Lambda H, S' \to \Lambda S, H \to \Lambda\}$

RE from a RG

Algorithm 8.1

Constructing an RE from a regular grammar
Input: A regular grammar $G = (V, \Sigma, S, P)$.
Output: A regular expression r over Σ, such that $\mathcal{L}(r) = \mathcal{L}(G)$.

Let V' be $V \cup \{S', H\}$, where S' is the new start variable
Add $S' \to \Lambda S$ and $H \to \Lambda$ to P
for each $A \to \Lambda \in P$ **do**
  Replace $A \to \Lambda$ by $A \to \Lambda H$ in P
[Now $G' = (V', \Sigma, S', P)$ is the first RGE and $\mathcal{L}(G') = \mathcal{L}(G)$.]
for each pair from P: $D \to r_1 E, D \to r_2 E$ **do**
  Replace the pair by $D \to (r_1 + r_2)E$ in P
while $V \neq \emptyset$ **do**
  Remove some B from V
  if no $B \to rB$ in P **then**
    Add $B \to \Lambda B$ to P
  for each triple from P: $A \to r_1 B, B \to r_2 B, B \to r_3 C$ **do**
    Add $A \to (r_1 (r_2)^* r_3)C$ to P
  for each pair from P: $D \to r_1 E, D \to r_2 E$ **do**
    Replace the pair by $D \to (r_1 + r_2)E$ in P
  Remove all productions using B from P
The only remaining productions are $S' \to rH$ and $H \to \Lambda$
return (r)

Example:

$$P = \{S \to aA,\, A \to aB,\, A \to aA,\; A \to bB,\, B \to aS,\, B \to bB,\; B \to \Lambda\}$$

$$P' = \{S \to aA,\, A \to aB,\, A \to aA,\; A \to bB,\, B \to aS,\, B \to bB,\; B \to \Lambda H,\, S' \to \Lambda S,\, H \to \Lambda\}$$

After 2nd For Each:

$$P' = \{S \to aA,\, A \to (a+b)B,\; A \to aA,\, B \to aS,\, B \to bB,\; B \to \Lambda H,\, S' \to \Lambda S,\, H \to \Lambda\}$$

Removing A from V

RE from a RG

Algorithm 8.1

Constructing an RE from a regular grammar
Input: A regular grammar $G = (V, \Sigma, S, P)$.
Output: A regular expression r over Σ, such that $\mathcal{L}(r) = \mathcal{L}(G)$.

Let V' be $V \cup \{S', H\}$, where S' is the new start variable
Add $S' \to \Lambda S$ and $H \to \Lambda$ to P
for each $A \to \Lambda \in P$ **do**
  Replace $A \to \Lambda$ by $A \to \Lambda H$ in P
[Now $G' = (V', \Sigma, S', P)$ is the first RGE and $\mathcal{L}(G') = \mathcal{L}(G)$.]
for each pair from P: $D \to r_1 E, D \to r_2 E$ **do**
  Replace the pair by $D \to (r_1 + r_2)E$ in P
while $V \neq \emptyset$ **do**
  Remove some B from V
  if no $B \to rB$ in P **then**
    Add $B \to \Lambda B$ to P
  for each triple from P: $A \to r_1 B, B \to r_2 B, B \to r_3 C$ **do**
    Add $A \to (r_1 (r_2)^* r_3)C$ to P
  for each pair from P: $D \to r_1 E, D \to r_2 E$ **do**
    Replace the pair by $D \to (r_1 + r_2)E$ in P
  Remove all productions using B from P
The only remaining productions are $S' \to rH$ and $H \to \Lambda$
return (r)

Example:
$P = \{S \to aA, A \to aB, A \to aA$
$A \to bB, B \to aS, B \to bB,$
$B \to \Lambda\}$

$P' = \{S \to aA, A \to aB, A \to aA$
$A \to bB, B \to aS, B \to bB,$
$B \to \Lambda H, S' \to \Lambda S, H \to \Lambda\}$

After 2nd For Each:
$P' = \{S \to aA, A \to (a+b)B,$
$A \to aA, B \to aS, B \to bB,$
$B \to \Lambda H, S' \to \Lambda S, H \to \Lambda\}$

Removing A from V
Triple:
$S \to aA, A \to aA, A \to (a+b)B$
Produces: $S \to (aa^*(a+b))B$
$P' = \{B \to aS, B \to bB, B \to \Lambda H,$
$S' \to \Lambda S, H \to \Lambda,$
$S \to (aa^*(a+b))B\}$

RE from a RG

Algorithm 8.1

Constructing an RE from a regular grammar
Input: A regular grammar $G = (V, \Sigma, S, P)$.
Output: A regular expression r over Σ, such that $\mathcal{L}(r) = \mathcal{L}(G)$.

Let V' be $V \cup \{S', H\}$, where S' is the new start variable
Add $S' \to \Lambda S$ and $H \to \Lambda$ to P
for each $A \to \Lambda \in P$ **do**
  Replace $A \to \Lambda$ by $A \to \Lambda H$ in P
[Now $G' = (V', \Sigma, S', P)$ is the first RGE and $\mathcal{L}(G') = \mathcal{L}(G)$.]
for each pair from P: $D \to r_1 E, D \to r_2 E$ **do**
  Replace the pair by $D \to (r_1 + r_2)E$ in P
while $V \neq \emptyset$ **do**
  Remove some B from V
  if no $B \to rB$ in P **then**
    Add $B \to \Lambda B$ to P
  for each triple from P: $A \to r_1 B, B \to r_2 B, B \to r_3 C$ **do**
    Add $A \to (r_1 (r_2)^* r_3)C$ to P
  for each pair from P: $D \to r_1 E, D \to r_2 E$ **do**
    Replace the pair by $D \to (r_1 + r_2)E$ in P
  Remove all productions using B from P
The only remaining productions are $S' \to rH$ and $H \to \Lambda$
return (r)

Example:
$$P = \{S \to aA, A \to aB, A \to aA \\ A \to bB, B \to aS, B \to bB, \\ B \to \Lambda\}$$
$$P' = \{S \to aA, A \to aB, A \to aA \\ A \to bB, B \to aS, B \to bB, \\ B \to \Lambda H, S' \to \Lambda S, H \to \Lambda\}$$

After 2nd For Each:
$$P' = \{S \to aA, A \to (a+b)B, \\ A \to aA, B \to aS, B \to bB, \\ B \to \Lambda H, S' \to \Lambda S, H \to \Lambda\}$$

Removing A from V
$$P' = \{B \to aS, B \to bB, B \to \Lambda H, \\ S' \to \Lambda S, H \to \Lambda, \\ S \to (aa^*(a+b))B\}$$

Removing B from V

RE from a RG

Algorithm 8.1

Constructing an RE from a regular grammar
Input: A regular grammar $G = (V, \Sigma, S, P)$.
Output: A regular expression r over Σ, such that $\mathcal{L}(r) = \mathcal{L}(G)$.

Let V' be $V \cup \{S', H\}$, where S' is the new start variable
Add $S' \to \Lambda S$ and $H \to \Lambda$ to P
for each $A \to \Lambda \in P$ **do**
  Replace $A \to \Lambda$ by $A \to \Lambda H$ in P
[Now $G' = (V', \Sigma, S', P)$ is the first RGE and $\mathcal{L}(G') = \mathcal{L}(G)$.]
for each pair from P: $D \to r_1 E, D \to r_2 E$ **do**
  Replace the pair by $D \to (r_1 + r_2)E$ in P
while $V \neq \emptyset$ **do**
  Remove some B from V
  if no $B \to rB$ in P **then**
    Add $B \to \Lambda B$ to P
  for each triple from P: $A \to r_1 B, B \to r_2 B, B \to r_3 C$ **do**
    Add $A \to (r_1(r_2)^* r_3)C$ to P
  for each pair from P: $D \to r_1 E, D \to r_2 E$ **do**
    Replace the pair by $D \to (r_1 + r_2)E$ in P
  Remove all productions using B from P
The only remaining productions are $S' \to rH$ and $H \to \Lambda$
return (r)

Example:
$P = \{S \to aA, A \to aB, A \to aA$
$A \to bB, B \to aS, B \to bB,$
$B \to \Lambda\}$

$P' = \{S \to aA, A \to aB, A \to aA$
$A \to bB, B \to aS, B \to bB,$
$B \to \Lambda H, S' \to \Lambda S, H \to \Lambda\}$

After 2nd For Each:
$P' = \{S \to aA, A \to (a+b)B,$
$A \to aA, B \to aS, B \to bB,$
$B \to \Lambda H, S' \to \Lambda S, H \to \Lambda\}$

Removing A from V
$P' = \{B \to aS, B \to bB, B \to \Lambda H,$
$S' \to \Lambda S, H \to \Lambda,$
$S \to (aa^*(a+b))B\}$

Removing B from V
Triple:
$S \to (aa^*(a+b))B, B \to bB, B \to aS$
Produces: $S \to ((aa^*(a+b))b^*a)S$

RE from a RG

Algorithm 8.1

Constructing an RE from a regular grammar
Input: A regular grammar $G = (V, \Sigma, S, P)$.
Output: A regular expression r over Σ, such that $\mathcal{L}(r) = \mathcal{L}(G)$.

Let V' be $V \cup \{S', H\}$, where S' is the new start variable
Add $S' \to \Lambda S$ and $H \to \Lambda$ to P
for each $A \to \Lambda \in P$ **do**
    Replace $A \to \Lambda$ by $A \to \Lambda H$ in P
[Now $G' = (V', \Sigma, S', P)$ is the first RGE and $\mathcal{L}(G') = \mathcal{L}(G)$.]
for each pair from P: $D \to r_1 E, D \to r_2 E$ **do**
    Replace the pair by $D \to (r_1 + r_2)E$ in P
while $V \neq \emptyset$ **do**
    Remove some B from V
    if no $B \to rB$ in P **then**
        Add $B \to \Lambda B$ to P
    for each triple from P: $A \to r_1 B, B \to r_2 B, B \to r_3 C$ **do**
        Add $A \to (r_1(r_2)^* r_3)C$ to P
    for each pair from P: $D \to r_1 E, D \to r_2 E$ **do**
        Replace the pair by $D \to (r_1 + r_2)E$ in P
    Remove all productions using B from P
The only remaining productions are $S' \to rH$ and $H \to \Lambda$
return (r)

Example:
$P = \{S \to aA, A \to aB, A \to aA$
$A \to bB, B \to aS, B \to bB,$
$B \to \Lambda\}$

$P' = \{S \to aA, A \to aB, A \to aA$
$A \to bB, B \to aS, B \to bB,$
$B \to \Lambda H, S' \to \Lambda S, H \to \Lambda\}$

After 2nd For Each:
$P' = \{S \to aA, A \to (a+b)B,$
$A \to aA, B \to aS, B \to bB,$
$B \to \Lambda H, S' \to \Lambda S, H \to \Lambda\}$

Removing A from V
$P' = \{B \to aS, B \to bB, B \to \Lambda H,$
$S' \to \Lambda S, H \to \Lambda,$
$S \to (aa^*(a+b))B\}$

Removing B from V
Triple:
$S \to (aa^*(a+b))B, B \to bB, B \to aS$
Produces: $S \to ((aa^*(a+b))b^*a)S$
Triple: $S \to (aa^*(a+b))B, B \to bB, B \to \Lambda H$
Produces: $S \to ((aa^*(a+b))b^*)H$

RE from a RG

Algorithm 8.1

Constructing an RE from a regular grammar
Input: A regular grammar $G = (V, \Sigma, S, P)$.
Output: A regular expression r over Σ, such that $\mathcal{L}(r) = \mathcal{L}(G)$.

Let V' be $V \cup \{S', H\}$, where S' is the new start variable
Add $S' \to \Lambda S$ and $H \to \Lambda$ to P
for each $A \to \Lambda \in P$ **do**
  Replace $A \to \Lambda$ by $A \to \Lambda H$ in P
[Now $G' = (V', \Sigma, S', P)$ is the first RGE and $\mathcal{L}(G') = \mathcal{L}(G)$.]
for each pair from P: $D \to r_1 E, D \to r_2 E$ **do**
  Replace the pair by $D \to (r_1 + r_2)E$ in P
while $V \neq \emptyset$ **do**
  Remove some B from V
  if no $B \to rB$ in P **then**
    Add $B \to \Lambda B$ to P
  for each triple from P: $A \to r_1 B, B \to r_2 B, B \to r_3 C$ **do**
    Add $A \to (r_1(r_2)^* r_3)C$ to P
  for each pair from P: $D \to r_1 E, D \to r_2 E$ **do**
    Replace the pair by $D \to (r_1 + r_2)E$ in P
  Remove all productions using B from P
The only remaining productions are $S' \to rH$ and $H \to \Lambda$
return (r)

Example:

$$P = \{S \to aA, A \to aB, A \to aA\\ A \to bB, B \to aS, B \to bB,\\ B \to \Lambda\}$$

$$P' = \{S \to aA, A \to aB, A \to aA\\ A \to bB, B \to aS, B \to bB,\\ B \to \Lambda H, S' \to \Lambda S, H \to \Lambda\}$$

After 2nd For Each:

$$P' = \{S \to aA, A \to (a+b)B,\\ A \to aA, B \to aS, B \to bB,\\ B \to \Lambda H, S' \to \Lambda S, H \to \Lambda\}$$

Removing A from V

$$P' = \{B \to aS, B \to bB, B \to \Lambda H,\\ S' \to \Lambda S, H \to \Lambda,\\ S \to (aa^*(a+b))B\}$$

Removing B from V

$$P' = \{S' \to \Lambda S, H \to \Lambda,\\ S \to ((aa^*(a+b))b^*a)S\\ S \to ((aa^*(a+b))b^*)H\}$$

RE from a RG

Algorithm 8.1

Constructing an RE from a regular grammar
Input: A regular grammar $G = (V, \Sigma, S, P)$.
Output: A regular expression r over Σ, such that $\mathcal{L}(r) = \mathcal{L}(G)$.

Let V' be $V \cup \{S', H\}$, where S' is the new start variable
Add $S' \to \Lambda S$ and $H \to \Lambda$ to P
for each $A \to \Lambda \in P$ **do**
  Replace $A \to \Lambda$ by $A \to \Lambda H$ in P
[Now $G' = (V', \Sigma, S', P)$ is the first RGE and $\mathcal{L}(G') = \mathcal{L}(G)$.]
for each pair from P: $D \to r_1 E, D \to r_2 E$ **do**
  Replace the pair by $D \to (r_1 + r_2)E$ in P
while $V \neq \emptyset$ **do**
  Remove some B from V
  if no $B \to rB$ in P **then**
    Add $B \to \Lambda B$ to P
  for each triple from P: $A \to r_1 B, B \to r_2 B, B \to r_3 C$ **do**
    Add $A \to (r_1(r_2)^* r_3)C$ to P
  for each pair from P: $D \to r_1 E, D \to r_2 E$ **do**
    Replace the pair by $D \to (r_1 + r_2)E$ in P
  Remove all productions using B from P
The only remaining productions are $S' \to rH$ and $H \to \Lambda$
return (r)

Example:
$P = \{S \to aA, A \to aB, A \to aA$
$A \to bB, B \to aS, B \to bB,$
$B \to \Lambda\}$

$P' = \{S \to aA, A \to aB, A \to aA$
$A \to bB, B \to aS, B \to bB,$
$B \to \Lambda H, S' \to \Lambda S, H \to \Lambda\}$

After 2nd For Each:
$P' = \{S \to aA, A \to (a+b)B,$
$A \to aA, B \to aS, B \to bB,$
$B \to \Lambda H, S' \to \Lambda S, H \to \Lambda\}$

Removing A from V
$P' = \{B \to aS, B \to bB, B \to \Lambda H,$
$S' \to \Lambda S, H \to \Lambda,$
$S \to (aa^*(a+b))B\}$

Removing B from V
$P' = \{S' \to \Lambda S, H \to \Lambda,$
$S \to ((aa^*(a+b))b^*a)S$
$S \to ((aa^*(a+b))b^*)H\}$

Removing S from V
$S' \to$
$(((aa^*(a+b))b^*a)^*)((aa^*(a+b))b^*)H$

RE from a RG

Algorithm 8.1

Constructing an RE from a regular grammar
Input: A regular grammar $G = (V, \Sigma, S, P)$.
Output: A regular expression r over Σ, such that $\mathcal{L}(r) = \mathcal{L}(G)$.

Let V' be $V \cup \{S', H\}$, where S' is the new start variable
Add $S' \to \Lambda S$ and $H \to \Lambda$ to P
for each $A \to \Lambda \in P$ **do**
  Replace $A \to \Lambda$ by $A \to \Lambda H$ in P
[Now $G' = (V', \Sigma, S', P)$ is the first RGE and $\mathcal{L}(G') = \mathcal{L}(G)$.]
for each pair from P: $D \to r_1 E, D \to r_2 E$ **do**
  Replace the pair by $D \to (r_1 + r_2)E$ in P
while $V \neq \emptyset$ **do**
  Remove some B from V
  if no $B \to rB$ in P **then**
    Add $B \to \Lambda B$ to P
  for each triple from P: $A \to r_1 B, B \to r_2 B, B \to r_3 C$ **do**
    Add $A \to (r_1(r_2)^* r_3)C$ to P
  for each pair from P: $D \to r_1 E, D \to r_2 E$ **do**
    Replace the pair by $D \to (r_1 + r_2)E$ in P
  Remove all productions using B from P
The only remaining productions are $S' \to rH$ and $H \to \Lambda$
return (r)

Claim: The loop invariant for the while loop is that the (currently modified) RGE has a derivation for a string x if and only if the original grammar G has a derivation for x.

RE from a RG

Algorithm 8.1

Constructing an RE from a regular grammar
Input: A regular grammar $G = (V, \Sigma, S, P)$.
Output: A regular expression r over Σ, such that $\mathcal{L}(r) = \mathcal{L}(G)$.

Let V' be $V \cup \{S', H\}$, where S' is the new start variable
Add $S' \to \Lambda S$ and $H \to \Lambda$ to P
for each $A \to \Lambda \in P$ **do**
  Replace $A \to \Lambda$ by $A \to \Lambda H$ in P
[Now $G' = (V', \Sigma, S', P)$ is the first RGE and $\mathcal{L}(G') = \mathcal{L}(G)$.]
for each pair from P: $D \to r_1 E, D \to r_2 E$ **do**
  Replace the pair by $D \to (r_1 + r_2)E$ in P
while $V \neq \emptyset$ **do**
  Remove some B from V
  if no $B \to rB$ in P **then**
    Add $B \to \Lambda B$ to P
  for each triple from P: $A \to r_1 B, B \to r_2 B, B \to r_3 C$ **do**
    Add $A \to (r_1(r_2)^* r_3)C$ to P
  for each pair from P: $D \to r_1 E, D \to r_2 E$ **do**
    Replace the pair by $D \to (r_1 + r_2)E$ in P
  Remove all productions using B from P
The only remaining productions are $S' \to rH$ and $H \to \Lambda$
return (r)

Claim: The loop invariant for the while loop is that the (currently modified) RGE has a derivation for a string x if and only if the original grammar G has a derivation for x. First, note that it holds when we arrive at the While loop.

RE from a RG

Algorithm 8.1

Constructing an RE from a regular grammar
Input: A regular grammar $G = (V, \Sigma, S, P)$.
Output: A regular expression r over Σ, such that $\mathcal{L}(r) = \mathcal{L}(G)$.

Let V' be $V \cup \{S', H\}$, where S' is the new start variable
Add $S' \to \Lambda S$ and $H \to \Lambda$ to P
for each $A \to \Lambda \in P$ **do**
  Replace $A \to \Lambda$ by $A \to \Lambda H$ in P
[Now $G' = (V', \Sigma, S', P)$ is the first RGE and $\mathcal{L}(G') = \mathcal{L}(G)$.]
for each pair from P: $D \to r_1 E, D \to r_2 E$ **do**
  Replace the pair by $D \to (r_1 + r_2)E$ in P
while $V \neq \emptyset$ **do**
  Remove some B from V
  if no $B \to rB$ in P **then**
    Add $B \to \Lambda B$ to P
  for each triple from P: $A \to r_1 B, B \to r_2 B, B \to r_3 C$ **do**
    Add $A \to (r_1(r_2)^* r_3)C$ to P
  for each pair from P: $D \to r_1 E, D \to r_2 E$ **do**
    Replace the pair by $D \to (r_1 + r_2)E$ in P
  Remove all productions using B from P
The only remaining productions are $S' \to rH$ and $H \to \Lambda$
return (r)

Claim: The loop invariant for the while loop is that the (currently modified) RGE has a derivation for a string x if and only if the original grammar G has a derivation for x.
First, note that it holds when we arrive at the While loop.
After each execution of the While loop, the invariant holds.

Deterministic Grammars

Consider the following grammar:
$P = \{S \to aA, A \to aB, A \to bA,\ A \to bB, B \to aS,\ B \to bB,\ B \to \Lambda\}$

Deterministic Grammars

Consider the following grammar:
$P = \{S \to aA, A \to aB, A \to bA,\ A \to bB, B \to aS,\ B \to bB,\ B \to \Lambda\}$

Generate: $x = aabaab$ and $y = aabaabb$

Deterministic Grammars

Consider the following grammar:
$P = \{S \to aA, A \to aB, A \to bA,\ A \to bB, B \to aS,\ B \to bB,\ B \to \Lambda\}$

Generate: $x = aabaab$ and $y = aabaabb$
$S \Rightarrow aA$

Deterministic Grammars

Consider the following grammar:
$P = \{S \to aA, A \to aB, A \to bA,\ A \to bB, B \to aS,\ B \to bB,\ B \to \Lambda\}$

Generate: $x = aabaab$ and $y = aabaabb$
$S \Rightarrow aA \Rightarrow aaB$

Deterministic Grammars

Consider the following grammar:
$P = \{S \rightarrow aA, A \rightarrow aB, A \rightarrow bA,\ A \rightarrow bB, B \rightarrow aS,\ B \rightarrow bB,\ B \rightarrow \Lambda\}$

Generate: $x = aabaab$ and $y = aabaabb$
$S \Rightarrow aA \Rightarrow aaB \Rightarrow aabB$

Deterministic Grammars

Consider the following grammar:
$P = \{S \to aA, A \to aB, A \to bA,\ A \to bB, B \to aS,\ B \to bB,\ B \to \Lambda\}$

Generate: $x = aabaab$ and $y = aabaabb$
$S \Rightarrow aA \Rightarrow aaB \Rightarrow aabB \Rightarrow aabaS$

Deterministic Grammars

Consider the following grammar:
$P = \{S \to aA, A \to aB, A \to bA,\ A \to bB, B \to aS,\ B \to bB,\ B \to \Lambda\}$

Generate: $x = aabaab$ and $y = aabaabb$
$S \Rightarrow aA \Rightarrow aaB \Rightarrow aabB \Rightarrow aabaS \Rightarrow aabaaA$

Deterministic Grammars

Consider the following grammar:
$P = \{S \to aA, A \to aB, A \to bA,\ A \to bB, B \to aS,\ B \to bB,\ B \to \Lambda\}$

Generate: $x = aabaab$ and $y = aabaabb$
$S \Rightarrow aA \Rightarrow aaB \Rightarrow aabB \Rightarrow aabaS \Rightarrow aabaaA \Rightarrow aabaabB \Rightarrow aabaab$

Deterministic Grammars

Consider the following grammar:
$P = \{S \to aA, A \to aB, A \to bA, A \to bB, B \to aS, B \to bB, B \to \Lambda\}$

Generate: $x = aabaab$ and $y = aabaabb$
$S \Rightarrow aA \Rightarrow aaB \Rightarrow aabB \Rightarrow aabaS \Rightarrow aabaaA \Rightarrow aabaabB \Rightarrow aabaab$
$\Rightarrow aabaabA \Rightarrow aabaabbB \Rightarrow aabaabb$

Deterministic Grammars

Consider the following grammar:
$P = \{S \to aA, A \to aB, A \to bA,\ A \to bB, B \to aS,\ B \to bB,\ B \to \Lambda\}$

Generate: $x = aabaab$ and $y = aabaabb$

$$S \Rightarrow aA \Rightarrow aaB \Rightarrow aabB \Rightarrow aabaS \Rightarrow aabaaA \Rightarrow aabaabB \Rightarrow aabaab$$
$$\Rightarrow aabaabA \Rightarrow aabaabbB \Rightarrow aabaabb$$

There are choices to make!

Deterministic Grammars

Consider the following grammar:
$P = \{S \to aA, A \to aB, A \to bA,\ A \to bB, B \to aS,\ B \to bB,\ B \to \Lambda\}$

Generate: $x = aabaab$ and $y = aabaabb$

$$S \Rightarrow aA \Rightarrow aaB \Rightarrow aabB \Rightarrow aabaS \Rightarrow aabaaA \Rightarrow aabaabB \Rightarrow aabaab$$
$$\Rightarrow aabaabA \Rightarrow aabaabbB \Rightarrow aabaabb$$

There are choices to make!
(In this case, only 1 correct choice.)

Deterministic Grammars

Consider the following grammar:
$P = \{S \to aA, A \to aB, A \to bA,\ A \to bB, B \to aS,\ B \to bB,\ B \to \Lambda\}$

Generate: $x = aabaab$ and $y = aabaabb$

$$S \Rightarrow aA \Rightarrow aaB \Rightarrow aabB \Rightarrow aabaS \Rightarrow aabaaA \Rightarrow aabaabB \Rightarrow aabaab$$
$$\Rightarrow aabaabA \Rightarrow aabaabbB \Rightarrow aabaabb$$

There are choices to make!
(In this case, only 1 correct choice.)
This requires "looking ahead". We call this non-determinism.

Deterministic Grammars

Consider the following grammar:
$P = \{S \to aA, A \to aB, A \to bA,\ A \to bB, B \to aS,\ B \to bB,\ B \to \Lambda\}$

Generate: $x = aabaab$ and $y = aabaabb$

$$S \Rightarrow aA \Rightarrow aaB \Rightarrow aabB \Rightarrow aabaS \Rightarrow aabaaA \Rightarrow aabaabB \Rightarrow aabaab$$
$$\Rightarrow aabaabA \Rightarrow aabaabbB \Rightarrow aabaabb$$

There are choices to make!
(In this case, only 1 correct choice.)
This requires "looking ahead". We call this non-determinism.

Deterministic Regular Grammars

A deterministic regular grammar G is a regular grammar that, for any $a \in \Sigma$ and any $A, B, C \in V$ with $B \neq C$, G does not have a pair of productions, $A \to aB$ and $A \to aC$.

Deterministic Grammars

Consider the following grammar:
$P = \{S \to aA, A \to aB, A \to bA,\ A \to bB, B \to aS,\ B \to bB,\ B \to \Lambda\}$

Generate: $x = aabaab$ and $y = aabaabb$

$$S \Rightarrow aA \Rightarrow aaB \Rightarrow aabB \Rightarrow aabaS \Rightarrow aabaaA \Rightarrow aabaabB \Rightarrow aabaab$$
$$\Rightarrow aabaabA \Rightarrow aabaabbB \Rightarrow aabaabb$$

There are choices to make!
(In this case, only 1 correct choice.)
This requires "looking ahead". We call this non-determinism.

Deterministic Regular Grammars

A deterministic regular grammar G is a regular grammar that, for any $a \in \Sigma$ and any $A, B, C \in V$ with $B \neq C$, G does not have a pair of productions, $A \to aB$ and $A \to aC$.

Lemma 8.4

If G is a regular grammar, then there exists a deterministic regular grammar G' such that $\mathcal{L}(G) = \mathcal{L}(G')$.