HW#1 CS 314 ANSWER SHEET

(0) $S \to A \mid S\#S \mid S@S$

$A \to C \mid CA$

$C \to a \mid b \mid c$

i) aa##bb

Cannot be derived, thus not in language.

Reason: Cannot achieve "##" i.e 2 #s side by side.

attempt:

$S \to S\#S$
$\to A\#S$
$\to \epsilon A\#S$
$\to aA\#S$
$\to aC\#S$
$\to aa\#S$
$\to aa\#A$
$\to aa\#CA$
$\to aa\#bA$
$\to aa\#bC$
$\to aa\#bb$. (Couldn't get 2 #s anyhow).

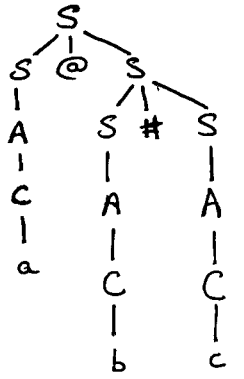
(ii) a@b#c

It is a language in G.

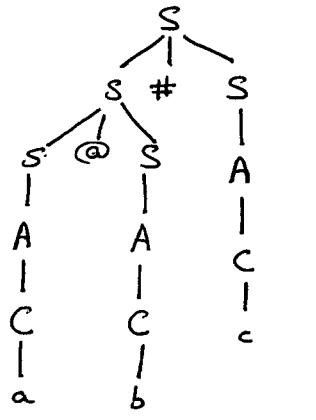
proof:

$S \Rightarrow S@S$
$\to S@S\#S$
$\to A@S\#S$
$\to C@S\#S$
$\to a@S\#S$
$\to a@A\#S$
$\to a@C\#S$
$\to a@b\#S$
$\to a@b\#A$
$\to a@b\#C$
$\to a@b\#c$

Parse tree 1



Parse tree 2



BUT, it is ambiguous, since it can be derived using multiple alternative ways.

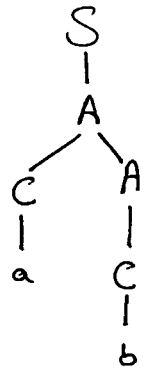
(iii) ab :

It is a language in G and is not ambiguous

Proof:

$S \rightarrow A$
$\rightarrow CA$
$\rightarrow aA$
$\rightarrow aC$
$\rightarrow ab$

Parse tree:



---

(1) Rewrite grammar G:

- no string is ambiguous
- '#' higher precedence than '@'
- '#' and '@' both are left associative

$S \rightarrow S@A \mid A$
$A \rightarrow A\#C \mid AC \mid C$
$C \rightarrow a \mid b \mid c$

⇒ lower in the tree means higher precedence (Thus '#' has higher precedence than '@')

⇒ Expansions from the left. (Left associativity).

⇒ For all strings above, there is only 1 derivation & parse tree. Thus unambiguous.

(2) For given grammar H:

For each production given:

- determine if it is in the language H
- explain why or why not.

i) a=0,6;

It is in language H:

Proof: (Bottom-up approach)

<var> = <Number>[, <Var>];

<var> = <value> [, <value>];

<Assignment>

<Statement> ⇒ Language.

(Top-down approach)

↳ Exact reverse of bottom-up.

ii) a=6, c, 1;

It's not in language H

Proof:

<Statement> → <Assignment>

<Assignment> → <var> = <value> [,<value>];

a = 6, c ... ??

We can have a maximum of 2 non-terminals / variables on the right of "=".

(iii) while (a){6 = 0; while (6) {}}:

It is in language H.

Proof: (Bottom-up approach)

while (<var>) { <var> = <Number>; while (<var>){ } }

while (<value>) {<var> = <value>; while (<value>) {{<Statement>}}}

while (<value>) {<var> = <value>; <While>}

while (<value>) { Assignment ⇒ statement }

while (<value>) { <Statement> }

• <while>

<Statement>

(iv) a=1; while (a) {a=1; while (a=0;}

It is not in Language:

proof:

a=1; while (a) {a=1; while (a=0;}

<var> = <number> ; while (<var>) {<var> = <number> while---}

<var> = <value> ; while (<var>) {<Assignment> <While>}

<var> = <value> ; while (<var>) {<Statement>}

<var> = <value> ; <while>

<Assignment> <Statement>

<Statement> <Statement>

There can't be an extension to Statement

or in other words

There can't be 2 or more Statements, side by side, according to the language unless inside '{ }'

(3) Write grammar for BNF in BNF :

⟨BNF⟩ ::= ⟨rule⟩ | ⟨rule⟩⟨BNF⟩

⟨rule⟩ ::= "⟨" ⟨rulename⟩ "⟩" "::=" ⟨expr⟩

⟨expr⟩ ::= ⟨list⟩ | ⟨list⟩ "|" ⟨expr⟩

⟨list⟩ ::= ⟨term⟩ | ⟨term⟩ ⟨list⟩

⟨term⟩ ::= ⟨text⟩ | "⟨" ⟨rulename⟩ "⟩"

⟨rulename⟩ ::= ⟨text⟩

⟨text⟩ ::= ⟨char⟩ ⟨text⟩ | ⟨char⟩

⟨char⟩ ::= ⟨letter⟩ | ⟨digit⟩ | ⟨symbol⟩

⟨letter⟩ ::= "A" | "B" | "C" | .... | "a" | "b" | "c" | ....

⟨digit⟩ ::= "0" | "1" | "2" | ....

⟨symbol⟩ ::= " | " | " _ " | " ; " | " : " | ....

There can be other ~~small~~ small bifurcations as well, but the logic to be followed is the same.