# CS 320: Homework 4

Due: November 21, 11:59pm

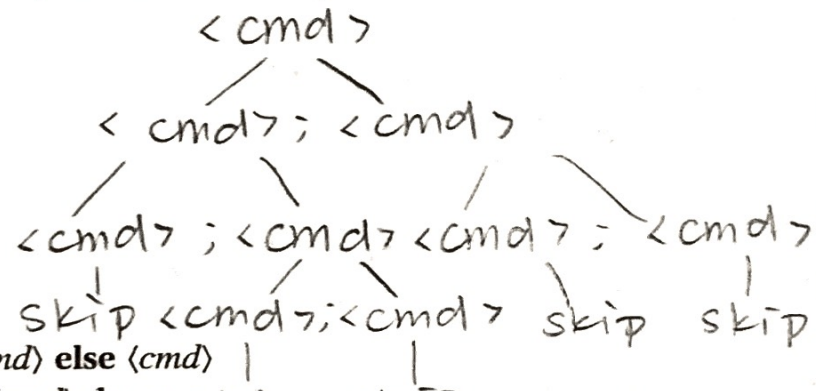
**Problem 1. Context Free Grammars (8 points)**

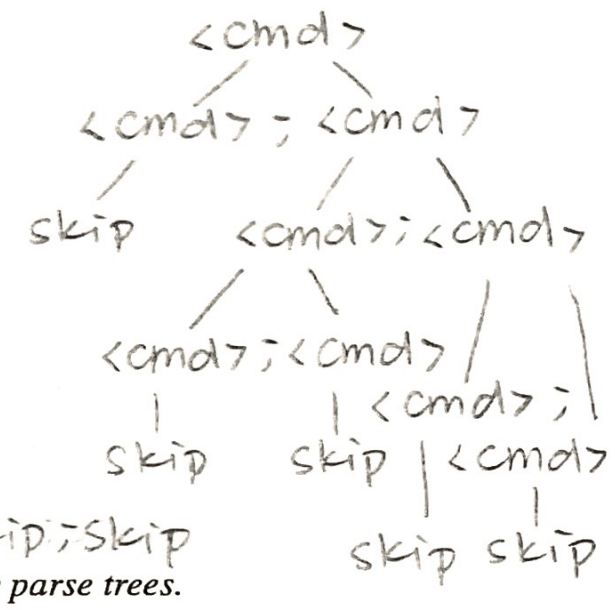
*Consider the following grammar defining a simple programming language. The square brackets used in the left rule are part of the concrete syntax. The initial symbol is* ***cmd****.*

⟨*expr*⟩ ::= ⟨*var*⟩ | ⟨*int*⟩

⟨*left*⟩ ::= ⟨*var*⟩
| ⟨*left*⟩[ ⟨*expr*⟩ ]

⟨*cmd*⟩ ::= **skip**
| ⟨*left*⟩ = ⟨*expr*⟩
| ⟨*cmd*⟩ ; ⟨*cmd*⟩
| **if** ⟨*expr*⟩ **then** ⟨*cmd*⟩ **else** ⟨*cmd*⟩
| **while** ⟨*expr*⟩ **do** ⟨*cmd*⟩ **done**



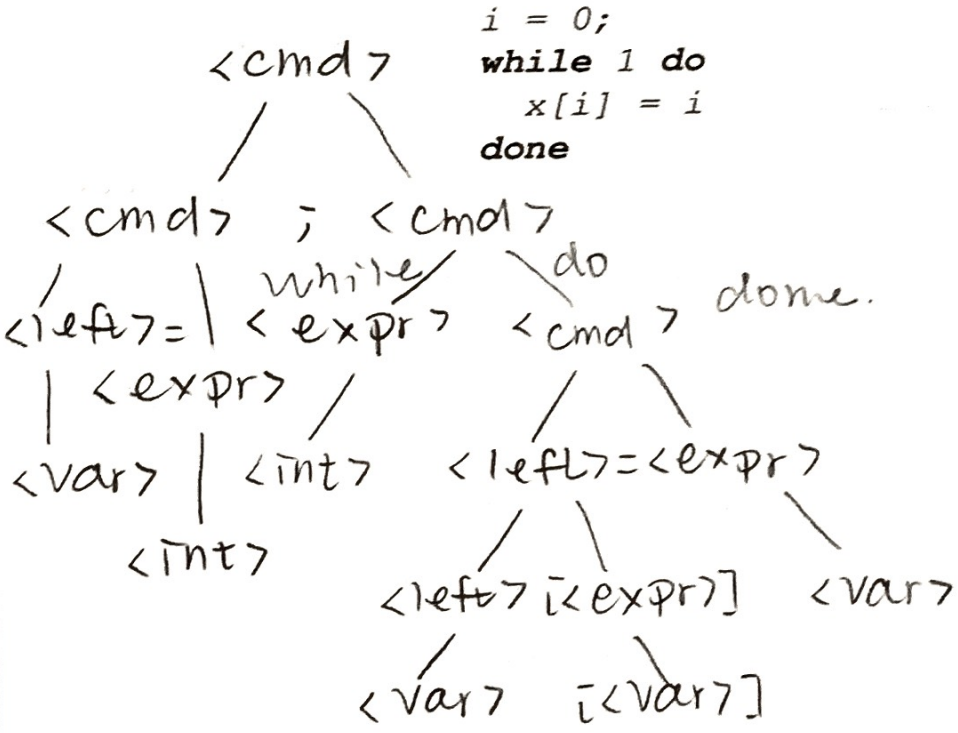




*The questions to answer are as follows.*

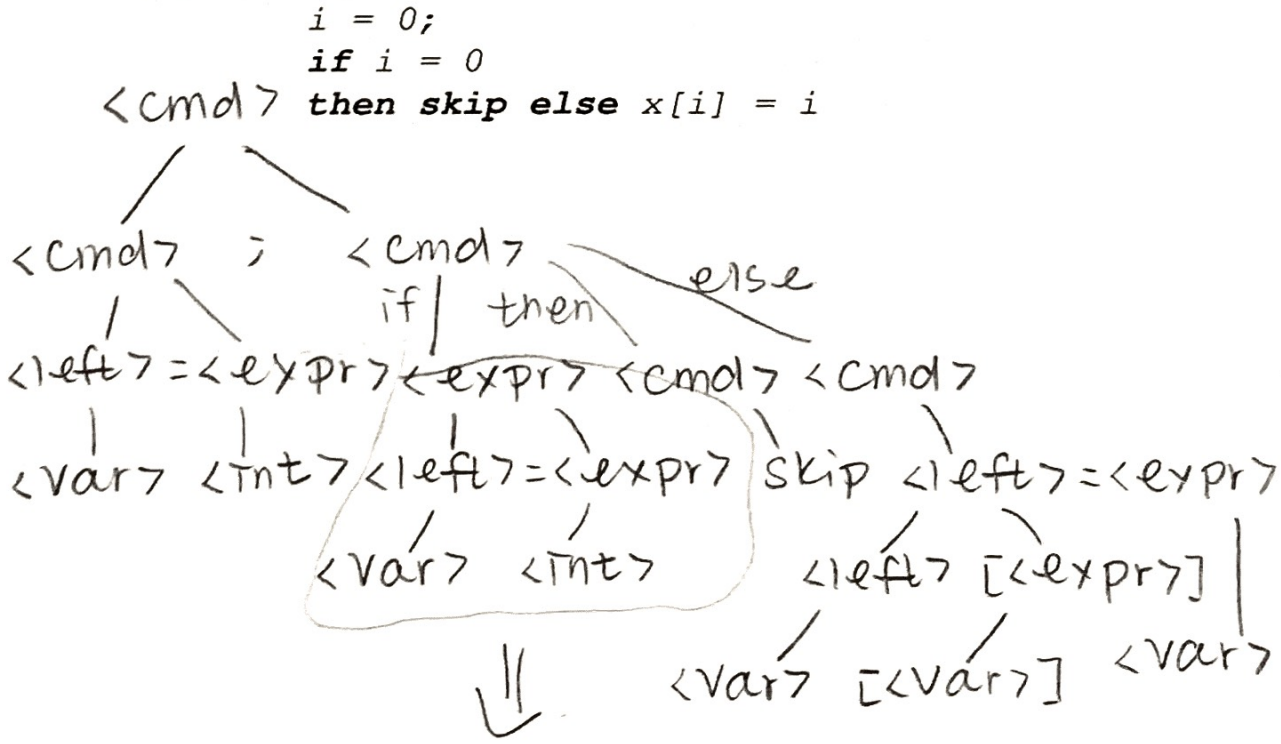
yes. string → skip;skip;skip;skip;skip

- *Is this grammar ambiguous? If so, show an example along with two of its possible parse trees.*
- *Is each of the following string accepted by this grammar? If yes, draw the parse tree, otherwise explain why not.*

```
i = 0;
while 1 do
  x[i] = i
done
```




```
i = 0;
if i = 0
then skip else x[i] = i
```



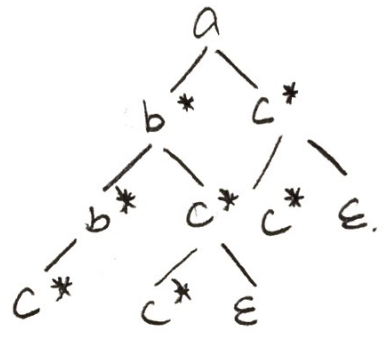
**Problem 2. Regular grammars and languages (4 points)**

*Consider the following regular grammar*

$\langle A \rangle ::= a\langle B \rangle \mid a\langle C \rangle$

$\langle B \rangle ::= b\langle C \rangle \mid b\langle B \rangle$

$\langle C \rangle ::= \epsilon \mid c\langle C \rangle$




ab*c*ε

*Please, provide an equivalent formulation in terms of regular expressions:*

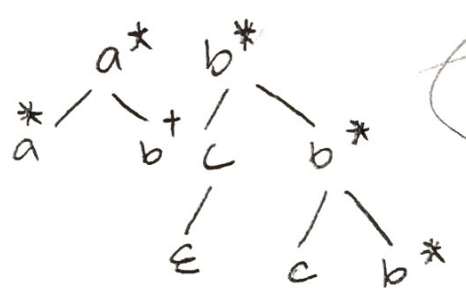
**Problem 3. Regular grammars and languages (4 points)**

*Consider the following regular grammar*

$\langle A \rangle ::= a\langle A \rangle \mid b\langle B \rangle$

$\langle B \rangle ::= c\langle C \rangle \mid b\langle B \rangle$

$\langle C \rangle ::= \epsilon$




a*bb*cε

*Please, provide an equivalent formulation in terms of regular expressions:*

**Problem 4. Regular grammars and languages (4 points)**

*Consider the following regular expression:*

$(ab)^* c^* \epsilon$

*Please, provide an equivalent formulation in terms of regular grammars:*

<A> ::= a<B> | ε

<B> ::= b<A> | b<C>

<C> ::= c<C> | ε

**Problem 5. Booleans and Integers (15 points)**
*Consider the following language of arithmetic expressions.*

$$
\begin{array}{lll}
\langle const \rangle & ::= & bool \\
& | & int \\
& | & \texttt{error} \\
\\
\langle expr \rangle & ::= & \langle const \rangle \\
& | & \textbf{add} \; ( \; \langle expr \rangle \; , \; \langle expr \rangle \; ) \\
& | & \textbf{eq} \; ( \; \langle expr \rangle \; , \; \langle expr \rangle \; )
\end{array}
$$

*Consider the following rules defining the operational semantics of arithmetic expressions. In this operational semantics a configuration is just an expression, which we will denote using the meta-variables $x, y, \ldots$. We use the notation $x \Rightarrow y$ to say that from the configuration/expression $x$ we can step to the configuration/expression $y$. Similarly, we use the notation $x \Rightarrow^k y$ to say that from the configuration/expression $x$ we can step, in $k$ steps, to the configuration/expression $y$.*

*Here $\mathbb{Z}$ is the set of all integers, and $\mathbb{B}$ is the set of all booleans. $+$, and $=$ are the usual mathematical notion of integer addition, and equality.*

$$\frac{}{x \Rightarrow^0 x}\ \textsc{Multi-Base} \qquad \frac{x \Rightarrow y \qquad y \Rightarrow^k z}{x \Rightarrow^{k+1} z}\ \textsc{Multi-Ind} \qquad \frac{x \Rightarrow x'}{\textbf{add}\ (x, y) \Rightarrow \textbf{add}\ (x', y)}\ \textsc{Add-Left}$$

$$\frac{x \in \mathbb{Z} \qquad y \Rightarrow y'}{\textbf{add}\ (x, y) \Rightarrow \textbf{add}\ (x, y')}\ \textsc{Add-Right} \qquad \frac{x \in \mathbb{Z} \qquad y \in \mathbb{Z}}{\textbf{add}\ (x, y) \Rightarrow (x + y)}\ \textsc{Add-Success}$$

$$\frac{x \in \mathbb{B} \cup \{\texttt{error}\}}{\textbf{add}\ (x, y) \Rightarrow \texttt{error}}\ \textsc{Add-Left-Error} \qquad \frac{x \in \mathbb{Z} \qquad y \in \mathbb{B} \cup \{\texttt{error}\}}{\textbf{add}\ (x, y) \Rightarrow \texttt{error}}\ \textsc{Add-Right-Error}$$

$$\frac{x \Rightarrow x'}{\textbf{eq}\ (x, y) \Rightarrow \textbf{eq}\ (x', y)}\ \textsc{Eq-Left} \qquad \frac{x \in \mathbb{Z} \qquad y \Rightarrow y'}{\textbf{eq}\ (x, y) \Rightarrow \textbf{eq}\ (x, y')}\ \textsc{Eq-Right}$$

$$\frac{x \in \mathbb{Z}}{\textbf{eq}\ (x, x) \Rightarrow\ \textbf{true}}\ \textsc{Eq-True} \qquad \frac{x, y \in \mathbb{Z} \qquad x \neq y}{\textbf{eq}\ (x, y) \Rightarrow\ \textbf{false}}\ \textsc{Eq-False}$$

$$\frac{x \in \mathbb{B} \cup \{\texttt{error}\}}{\textbf{eq}\ (x, y) \Rightarrow \texttt{error}}\ \textsc{Eq-Left-Error} \qquad \frac{x \in \mathbb{Z} \qquad y \in \mathbb{B} \cup \{\texttt{error}\}}{\textbf{eq}\ (x, y) \Rightarrow \texttt{error}}\ \textsc{Eq-Right-Error}$$

*Prove the following judgments by drawing their derivation trees.*

- $\textbf{add}\ (1, \textbf{add}\ (2,3)) \Rightarrow^2 6$
- $\textbf{eq}\ (\textbf{add}\ (1,2)), \textbf{add}\ (2,1)) \Rightarrow^3\ \textbf{true}$
- $\textbf{add}\ (\textbf{add}\ (1,2), \textbf{eq}\ (1,2)) \Rightarrow^3 \texttt{error}$

add(1,5) => (1+5)

add(1, add(2,3)) => add(1, (2+3))

add(1, add(2,3)) =>$^{2}$6

6 =>$^{0}$6

add(1, 5) =>$^{1}$6

eq(3,3) => true

eq(3, add(2,1)) => eq(3, (2+1))

eq(add(1,2), add(2,1)) => eq((1+2), add(2,1))

eq(add(1,2), add(2,1)) =>$^{3}$ true

true =>$^{0}$ true

eq(3,3) =>$^{1}$ true

eq(3, add(2,1)) =>$^{2}$ true

B

add(3, false) => error

add(3, eq(1,2)) => add(3, false)

add(add(1,2), eq(1,2)) => add((1+2), eq(1,2))

add(add(1,2), eq(1,2)) =>$^{3}$ error

error =>$^{0}$ error

add(3, false) =>$^{1}$ error

add(3, eq(1,2)) =>$^{2}$ error

**Problem 6. Stack Language (15 points)** *Consider the following stack language comprised of commands operating on stacks.*

$\langle const \rangle$ ::= *int*

$\langle cmd \rangle$ ::= **push** $\langle const \rangle$ | **pop**
| **add**
| **sub**
| **if** $\langle cmds \rangle$ **else** $\langle cmds \rangle$ **end**
| **quit**

$\langle cmds \rangle$ ::= $\langle cmd \rangle$
| $\langle cmd \rangle$ ; $\langle cmds \rangle$

$\langle stack \rangle$ ::= []
| $\langle const \rangle$ :: $\langle stack \rangle$

*We use the meta-variable c to denote a configuration. A configuration can be a pair $(p/s)$ comprising a program p and a stack s, or an* `error`*. The judgment $c \Rightarrow c'$ indicates that from the configuration c we can step to the configuration $c'$ in a single step. This is used to define the judgment $c \Rightarrow^k c'$ indicating that from the configuration c we can step to the configuration $c'$ in k steps. The following specify the rules in detail.*

$$\frac{}{c \Rightarrow^0 c}\ \textsc{Multi-Base} \qquad \frac{c \Rightarrow c' \qquad c' \Rightarrow^k c''}{c \Rightarrow^{k+1} c''}\ \textsc{Multi-Ind} \qquad \frac{}{(\textbf{push}\ v; p\ /\ s) \Rightarrow (p\ /\ v :: s)}\ \textsc{Push}$$

$$\frac{}{(\textbf{pop}\ ; p\ /\ v :: s) \Rightarrow (p\ /\ s)}\ \textsc{Pop} \qquad \frac{}{(\textbf{pop}\ ; p\ /\ []) \Rightarrow \texttt{error}}\ \textsc{Pop-Error}$$

$$\frac{}{(\textbf{add}\ ; p\ /\ v_1 :: v_2 :: s) \Rightarrow (p\ /\ (v_2 + v_1) :: s)}\ \textsc{Add} \qquad \frac{}{(\textbf{add}\ ; p\ /\ v_1 :: []) \Rightarrow \texttt{error}}\ \textsc{Add-Error1}$$

$$\frac{}{(\textbf{add}\ ; p\ /\ []) \Rightarrow \texttt{error}}\ \textsc{Add-Error2}$$

$$\frac{}{(\textbf{sub}\ ; p\ /\ v_1 :: v_2 :: s) \Rightarrow (p\ /\ (v_2 - v_1) :: s)}\ \textsc{Sub} \qquad \frac{}{(\textbf{sub}\ ; p\ /\ v_1 :: []) \Rightarrow \texttt{error}}\ \textsc{Sub-Error1}$$

$$\frac{}{(\textbf{sub}\ ; p\ /\ []) \Rightarrow \texttt{error}}\ \textsc{Sub-Error2}$$

$$\frac{v > 0}{(\ \textbf{if}\ p_1\ \textbf{else}\ p_2\ \textbf{end}\ ; p\ /\ v :: s) \Rightarrow (p_1; p\ /\ s)}\ \textsc{If-True}$$

$$\frac{v \leq 0}{(\ \textbf{if}\ p_1\ \textbf{else}\ p_2\ \textbf{end}\ ; p\ /\ v :: s) \Rightarrow (p_2; p\ /\ s)}\ \textsc{If-False} \qquad \frac{}{(\ \textbf{if}\ p_1\ \textbf{else}\ p_2\ \textbf{end}\ ; p\ /\ []) \Rightarrow \texttt{error}}\ \textsc{If-Error}$$

*Prove the following judgments by drawing their derivation trees.*

- (**push** 1; **push** 2; **add** ; **quit** / []) $\Rightarrow^3$ (**quit** / 3 :: [])

- **(push** 1;**push** 2;**sub** ;**quit** / []) $\Rightarrow^3$ (**quit** / $-1 :: []$)
- **(push** 1;**push** 2;**sub** ; **if push** 1 **else pop end** ;**quit** / []) $\Rightarrow^5$ `error`

①.

(quit/3::i]) => 0 (quit/3::i])

(add;quit/2::1::i]) => (quit/(2+1)::i]) | 1 (quit/3::i])

(push 2;add;quit/1::i]) => (add;quit/2::1::i]) (add;quit/2::1::i]) =>

(push 1;push 2;add;quit/i]) => (push 2;add;quit/1::i]) (push 2;add;quit/1::i]) =>

(push 1;push 2;add;quit/i]) => 3 (quit/3::i]) | 2 (quit/3::i])

②.

(quit/-1::i]) => 0 (quit/-1::i])

(sub;quit/2::1::i]) => (quit/(1-2)::i]) | 1 (quit/-1::i])

(push 2;sub;quit/1::i]) => (sub;quit/2::1::i]) (sub;quit/2::1::i]) =>

(push 1;push 2;sub;quit/i]) => (push 2;sub;quit/1::i]) (push 2;sub;quit/1::i]) =>

(push 1;push 2;sub;quit/i]) => 3 (quit/-1::i]) | 2 (quit/-1::i])

$$\frac{}{(\mathtt{pop};\mathtt{quit}/i] \Rightarrow \mathtt{error}} \quad ⑤$$

$$\frac{}{-1 \leq 0}$$

$$(\mathtt{if\ push\ 1\ else\ pop\ end};\mathtt{quit}/-1::i]) \Rightarrow (\mathtt{pop};\mathtt{quit}/i] \quad ④$$

$$(\mathtt{sub};\mathtt{if\ push\ 1\ else\ pop\ end};\mathtt{quit}/2::1::i]) \Rightarrow (\mathtt{if\ push\ 1\ else\ pop\ end};\mathtt{quit}/(1-2)::i] \quad ③$$

$$(\mathtt{push\ 2};\mathtt{sub};\mathtt{if\ push\ 1\ else\ pop\ end};\mathtt{quit}/1::i] \Rightarrow (\mathtt{sub};\mathtt{if\ push\ 1\ else\ pop\ end};\mathtt{quit}/2::1::i] \quad ②$$

$$(\mathtt{push\ 1};\mathtt{push\ 2};\mathtt{sub};\mathtt{if\ push\ 1\ else\ pop\ end};\mathtt{quit}/i] \Rightarrow (\mathtt{push\ 2};\mathtt{sub};\mathtt{if\ push\ 1\ else\ pop\ end};\mathtt{quit}/1::i] \quad ①$$

$$(\mathtt{push\ 1};\mathtt{push\ 2};\mathtt{sub};\mathtt{if\ push\ 1\ else\ pop\ end};\mathtt{quit}/i] \Rightarrow^5 \mathtt{error}$$

⑤ $\mathtt{error} \Rightarrow^0 \mathtt{error}$

④ $(\mathtt{pop};\mathtt{quit}/i] \Rightarrow^1 \mathtt{error}$

③ $(\mathtt{if\ push\ 1\ else\ pop\ end};\mathtt{quit}/-1::i]) \Rightarrow^2 \mathtt{error}$

② $(\mathtt{sub};\mathtt{if\ push\ 1\ else\ pop\ end};\mathtt{quit}/2::1::i]) \Rightarrow^3 \mathtt{error}$

① $(\mathtt{push\ 2};\mathtt{sub};\mathtt{if\ push\ 1\ else\ pop\ end};\mathtt{quit}/1::i]) \Rightarrow^4 \mathtt{error}$

$\ldots \Rightarrow^5 \mathtt{error}$