Matthew O'Connor Student # 50000250 Homework one Cse 305

Q1. Assuming right braces, the 'end' keyword, and distinct closing syntaxes all share the same functionality, then there is really not very much difference between the three. The only real difference is readability vs coding time. Having to remember and type out separate syntaxes would take the longest amount of time, but would improve readability ad it distinguished what was going on. Right braces take the least amount of time but when many stack up, it can get confusing as to what ends where. End statements would be somewhere in-between.

Q2. Type declaration statements create a rigid syntax, whereas not having type syntax statements usually creates a dynamic syntax, so this question boils down to a more rigid versus a more dynamic syntax. A rigid syntax makes compiling faster, as things do not need to be constantly re-interpreted. Dynamic syntax needs to be re- interpreted and therefore makes a program run slower. A rigid syntax also keeps a programmer from being confused and accidentally using an integer as a string or making any similar errors. The disadvantage of rigid type statements would be less flexibility. Dynamic syntax allows a programmer to do more with variables as he has less constraints, and can lead to having some good tricks, and it can allow for less casting. However, more freedom can cause more errors.

Q3

Part A

End: aaaX+aaXX

1. <stmt>

2. <stmt><stmt>x

3. <stmt><stmt>+<stmt>x

4. a<stmt>+<stmt>x

5. a<stmt><stmt>x+<stmt>x

6.) aa<stmt>x+<stmt>x

7.) aaax+<stmt>x

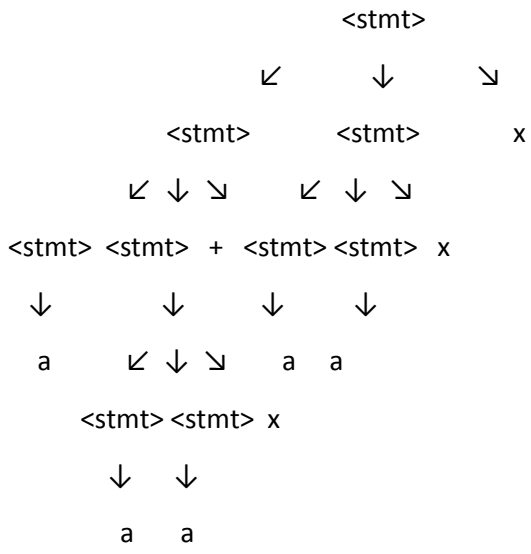
8.) aaax+<stmt><stmt>xx

9.)aaax+a<stmt>xx

10.)aaax+aaxx

Part b.)

.



Part c.) A context free language is the set of all strings generated by its grammar. So the language is generate able strings of a's +'s and x's

Part d.) The grammar is ambiguous since a <stmt> can be multiple things and therefore there are multiple orders in which a <stmt> can be broken down making the language ambiguous. Also the order in which x's and +'s is ambiguous. You could also break A's down before statements and vice versa.

Q4.) Since <ST> can be <MatchedST> this grammar can produce and endless pile of elses.

Such as <MatchedST> => IF Be than <MatchedST => if be than<matchedST=>….> >else<ST=MatchedST>. Since you can stick a Matched ST in all the else slots in place of a regular ST, you can pile elses upon elses and make things very ambiguous. Also |other is very ambiguous and not well defined.

Q5.) The grammar we discussed in class has a left associative sum operator, meeting the requirements for part B, and I will put it here as reference since some small changes will meet the requirements for the question.

<assign> → <id> = <expr>

<expr> → <expr> + <term> | <term>

<term> → <term> * <factor> | <factor>

<factor> → (<expr>) | <id>

<id> → A | B | C

a.) Simply moving the + and * operators from the grammar we discussed in class changes the precedence

<assign> → <id> = <expr>

<expr> → <expr> *<term> | <term>

<term> → <term> + <factor> | <factor>

<factor> → (<expr>) | <id>

b.) The original grammar and the grammar I used in part a.) meet this requirement.

c.) If we flip <expr> and term and term and factor we get right associative.

<assign> → <id> = <expr>

<expr> → <term> +<expr> | <term>

<term> → <factor> * <term> | <factor>

<factor> → (<expr>) | <id>

d.) Simply adding a negative id <nid> gives us this grammar

<assign> → <id> = <expr>

<expr> → <expr> + <term> | <term>

<term> → <term> * <factor> | <factor>

<factor> → (<expr>) | <id>

<id> → A | B | C

<nid> → -A | -B| -C

6. | ?- consult('family.P').

[Compiling ./family]

[family compiled, cpu time used: 0.0100 seconds]

[family loaded]

yes

| ?- father(david, sarah).

no

| ?- father(F, C).

F = david

C = kian.

no

| ?- mother(anna, X).

X = kian.

No

7.

timberlake {~/cse305} > vi hw1sum.sml

timberlake {~/cse305} > clear

timberlake {~/cse305} > sml

Standard ML of New Jersey v110.69 [built: Thu May 28 09:54:29 2009]

- use "hw1sum.sml";

[opening hw1sum.sml]

val sum = fn : int list -> int

val it = () : unit

- sum([1,4,8,17]);

val it = 30 : int

- val z = [1,3,5,7,9,11,13];

val z = [1,3,5,7,9,11,13] : int list

- sum(z);

val it = 49 : int

-