Bryan Kline

CS326

Homework 1

LaTex (TeXstudio)

09/15/2016

1. Why are the front end and the back end of a compiler usually implemented as separate passes?

   Dividing the process of generating a program, from high level language to machine code, into two separate passes allows for portability on the front end and speed on the back end. Separating the back end makes it machine independent, so that there can be high level languages that can be used to write code for many different machines, and then the back end implements it for a specific machine.

2. For each of the following languages, write a regular expression that describes the language.

   (a) The set of strings of length three or more, over alphabet {a, b}.

   $(a|b)(a|b)(a|b)(a|b)^*$

   (b) The set of natural numbers divisible by 25.

   $digit = 1|2|3|4|5|6|7|8|9|0$
   $(digit)^*$ $00|25|50|75$

   (c) The set of strings that consist of an odd number of as, over alphabet {a, b}.

   $(a(aa)^*)|\epsilon$

   (d) The set of strings over alphabet {a, b} that begin with at least two as, and end with at least two bs.

   $aa(a|b)^*bb$

3. For each of the following languages, write a grammar that describes the language.

   (a) The set of strings over alphabet {a, b} that begin with at least two as, and end with at least two bs (same language as above).

   $Expr \rightarrow aaSbb|\epsilon$
   $S \rightarrow idS|\epsilon$
   $id \rightarrow a|b$

   (b) The set of strings that consist of an even number of a's, over alphabet {a}.

   $Expr \rightarrow aaS|\epsilon$
   $S \rightarrow aaS|\epsilon$

   (c) The set of strings of parentheses ( ), brackets [ ], and braces { } that are properly nested. For instance, ( ) [( )] is properly nested, while ([ )] is not.

   $S \rightarrow (SS)|[SS]|\{SS\}|SS|\epsilon$

4.Consider the following grammar for Scheme:

$expr \rightarrow ATOM|list$
$list \rightarrow (exprs)$
$exprs \rightarrow expr\ exprs|\epsilon$

Using this grammar, show a parse tree for the expression $(lambda(a)(*aa))$. Does the language described by this grammar contain a finite number of strings? If so, why? If not, why not?

Blank

5. Consider the following grammar for a declaration list:

$decl\_list \rightarrow decl\ ;\ decl\_list | \epsilon$
$decl \rightarrow specifier\ type\ name\_list$
$specifier \rightarrow const | static | \epsilon$
$type \rightarrow double | int$
$name\_list \rightarrow name | name\ ,\ name\_list$
$name \rightarrow id\ args$
$args \rightarrow (decl\_list) | \epsilon$

(a) Indicate whether each of the following strings belongs to the language described by the grammar.

i	`int a (int b);`	Cannot be described.
ii	`int c (int d (int e;););`	Can be described.
iii	`double f, g (static int h;);`	Can be described.
iv	`static int i; const double j;`	Can be described.

A tree is provided to show whether each string belongs to the language.
See following page.

(b) Show a leftmost derivation of the string `static int f();` under this grammar.

A tree is provided to show whether the string belongs to the language.
See following page.

(c) Indicate whether each of the following strings belongs to the language described by the grammar.

NOTE: I'm assuming that the grammar for part (c) is independent of the previous parts of the question and so won't necessarily have to produce valid strings for those parts.

$decl\_list \rightarrow decl\ ;\ decl\_list | decl\ ,\ decl\_list | \epsilon$
$decl \rightarrow specifier\ type\ name\_list$
$specifier \rightarrow const | static | \epsilon$
$type \rightarrow double | int$
$name\_list \rightarrow name | name\ ,\ name\_list$
$name \rightarrow id\ args$
$args \rightarrow (decl | decl\ ,\ decl\_list) | \epsilon$