Amirkabir University of Technology
(Tehran Polytechnic)

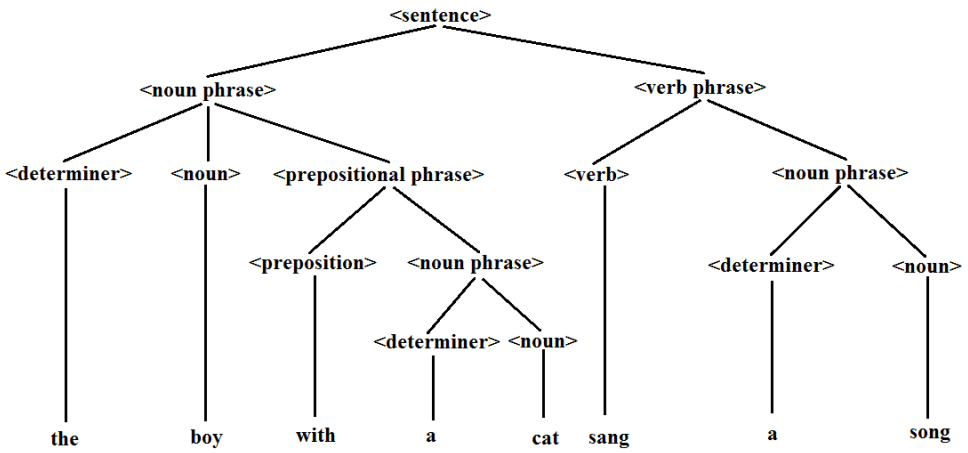
Topic
First Exercise of Programming Languages

Author
Amirmohammad Nazari

Professor
Mehran S. Fallah

Answer to question two on page eight

We can write two different derivation of the sentence as next derivations.
<sentence> ::= <noun phrase> <verb phrase>::= <determiner> <noun> <prepositional phrase> <verb phrase>::= the <noun> <prepositional phrase> <verb phrase>::= the boy <prepositional phrase> <verb phrase>::= the boy <preposition> <noun phrase> <verb phrase>::= the boy with <noun phrase> <verb phrase>::= the boy with <determiner> <noun> <verb phrase>::= the boy with a <noun> <verb phrase>::= the boy with a cat <verb phrase>::= the boy with a cat <verb> <noun phrase>::= the boy with a cat sang <noun phrase>::= the boy with a cat sang <determiner> <noun>::= the boy with a cat sang a <noun>::= the boy with a cat sang a song
<sentence> ::= <noun phrase> <verb phrase>::= <noun phrase> <verb> <noun phrase>::= <noun phrase> <verb> <determiner> <noun>::= <noun phrase> <verb> <determiner> song::= <noun phrase> <verb> a song::= <noun phrase> sang a song::= <determiner> <noun> <prepositional phrase> sang a song::= <determiner> <noun> <preposition> <noun phrase> sang a song::= <determiner> <noun> <preposition> <determiner> <noun> sang a song::= <determiner> <noun> <preposition> <determiner> cat sang a song::= <determiner> <noun> <preposition> a cat sang a song::= <determiner> <noun> with a cat sang a song::= <determiner> boy with a cat sang a song::= the boy with a cat sang a song
We can draw derivation tree of two different derivations as next derivation tree.



Answer to question three on page eight

We can write meaning of linguistics, semiotics, grammar, syntax, semantics and pragmatics as next definitions.
Linguistics: the scientific study of language and its structure, including the study of morphology, syntax, phonetics, and semantics.
Semiotics: the study of signs and symbols and their use or interpretation.
Grammar: the whole system and structure of a language or of languages in general, usually taken as consisting of syntax and morphology and sometimes also phonology and semantics.
Syntax: the arrangement of words and phrases to create well-formed sentences in a language.
Semantics: the branch of linguistics and logic concerned with meaning.
Pragmatics: the branch of linguistics dealing with language in use and the contexts in which it is used, including such matters as deixis, taking turns in conversation, text organization, presupposition, and implicature.

Answer to question five on page eight

We can derive the statement as next derivation.
<sentence> ::= a<thing>bc ::= ab<thing>c ::= ab<other>bcc ::= a<other>bbcc ::= aa<thing>bbcc ::= aab<thing>bcc ::= aabb<thing>cc

::= aabb<other>bccc ::= aab<other>bbccc ::= aa<other>bbbccc ::= aaabbbccc

Answer to question six on page eight

Answer to part a

The grammar is ambiguous because there are two different derivation or two different derivation tree for $\epsilon\epsilon\epsilon$.
<String>= <String> <String>= ϵ <String>= ϵ <String> <String>= ϵ ϵ <String>=$\epsilon\epsilon\epsilon$
<String>= <String> <String>= <String> ϵ= <String> <String> ϵ= <String> ϵ ϵ=$\epsilon\epsilon\epsilon$

Answer to part b

The grammar is not ambiguous. I tried to find at least a sentence which has two different derivation tree but I could not find.

Answer to question seven on page eight

Answer to part a

The grammar describes $\{a^n b^n | n = 1, 2, ...\}$.

Answer to part b

The grammar describes $\{ww^R | w \in \{a, b\}^*\}$.

Answer to part c

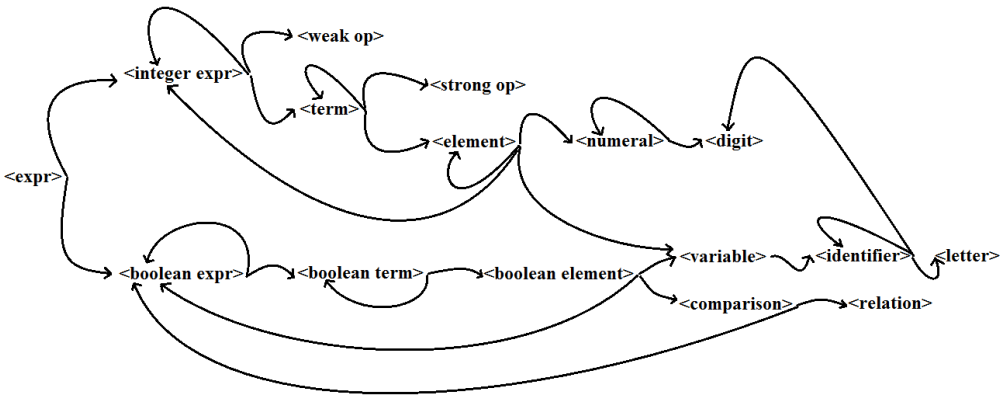
The grammar describes $\{w | w \in \{a, b\}^+, w$ contains an equal number of occurrences of characters a and b$\}$.

Answer to question nine on page eight

We can use proof by construction in order to prove that concatenation of two regular grammars is a regular grammar. Regular grammars have DFA. You can merge two DFA in order to obtain another DFA for concatenation of two regular grammars. So the concatenation of two regular grammars is a regular grammar.
We can define English characters with regular grammars. Therefore, English grammar is a regular grammar since English grammar is concatenation of different characters.
We can rewrite the grammar as next grammar.
<sentence> ::= a boy <verb phrase> .
<noun phrase> ::= a boy | a boy <prepositional phrase>
<verb phrase> ::= saw | saw <noun phrase> | saw a boy <prepositional phrase>
<prepositional phrase> ::= by <noun phrase>
<noun> ::= boy
<determiner> ::= a
<verb> ::= saw
<preposition> ::= by

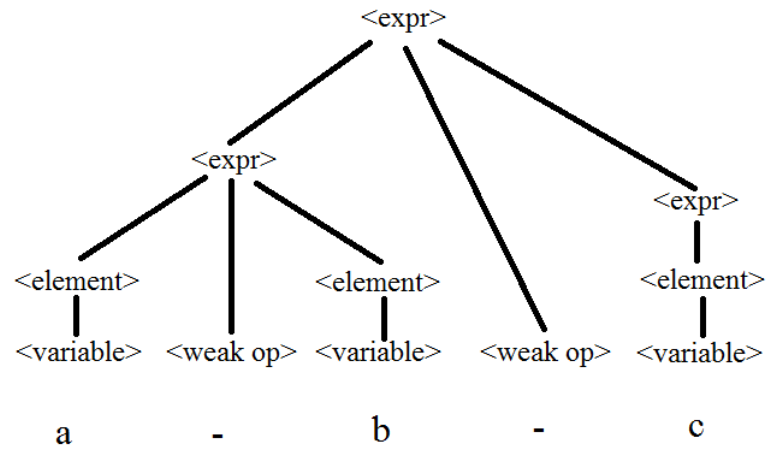
Answer to question one on page sixteen

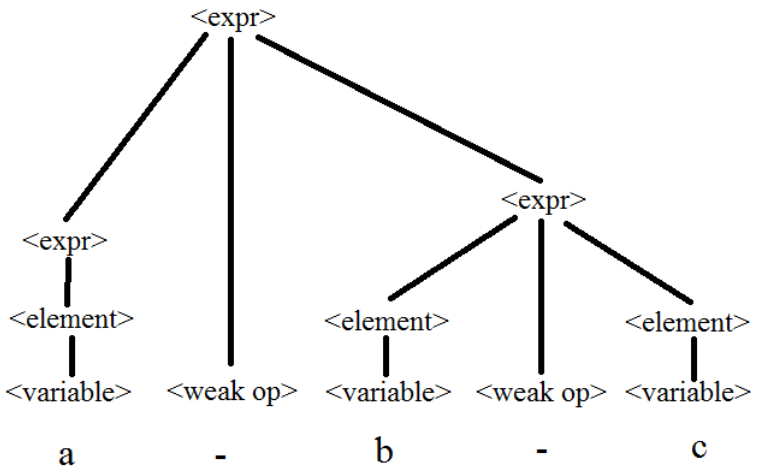
We can draw a dependency graph for the nonterminal <expr> in the BNF definition of Wren.



Answer to question two on page sixteen

The derivation trees demonstrate bellow.





The Wren avoid ambiguity by begin and end, and parentheses.

Answer to question three on page sixteen

I am going to mention each error consecutively.

"was" should be "is" at first line. It is context-free error.
";" should be ":" at third line. It is context-free error.
b is defined two times at line two and line three. It is context-sensitive error.
$\neq$ should be "¡¿" at sixth line. It is context-free error.
"p:= (a+1)" has type problem since the right side is an integer and the left side is a boolean. It is context-sensitive error.
q is uninitialized. It is context-sensitive error.
"b $\neq$ 0" is an error since we didn't assign a value to "b". It is semantic error.
"write q" is an error since we didn't assign a value to "q". It is semantic error.

Answer to question five on page sixteen

Answer to part a

If we scrutinize the grammar then we will understand priorities based on the grammar. Subtraction has first priority. Summation has second priority. Multiplication has third priority.

Answer to part b

Multiplication is right recursive so the order is right-to-left. Subtraction is right recursive so the order is right-to-left. Summation is left recursive so the order is left-to-right.

Answer to part c

In fact, parentheses only have effect on priority of subtraction. They clarify priority of execution for consecutive subtraction.

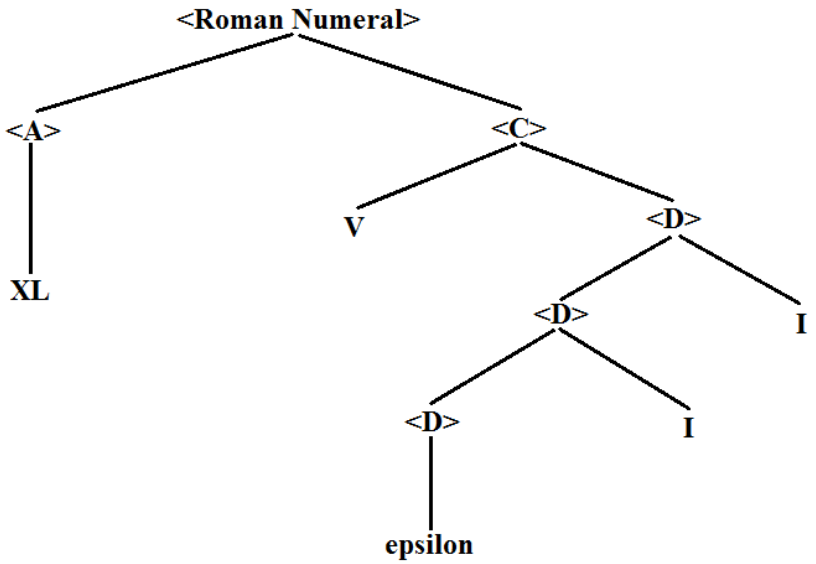
Answer to question seven on page sixteen

Right recursive grammars have form like A$\Rightarrow \alpha$ A. In addition, left recursive grammars have form like A$\Rightarrow$ Aα. As a result, associativity of the binary

operations is right-to-left in right recursive grammars. Also, associativity of the binary operations is left-to-right in left recursive grammars.

Answer to question eight on page sixteen

<Roman Numeral>::= <A> <C>
<A>::= <B> | XL | L <B> | XC
<B>::= ϵ | <B> X
<C>::= <D> | IV | V <D> | IX
<D>::= ϵ | <D> I

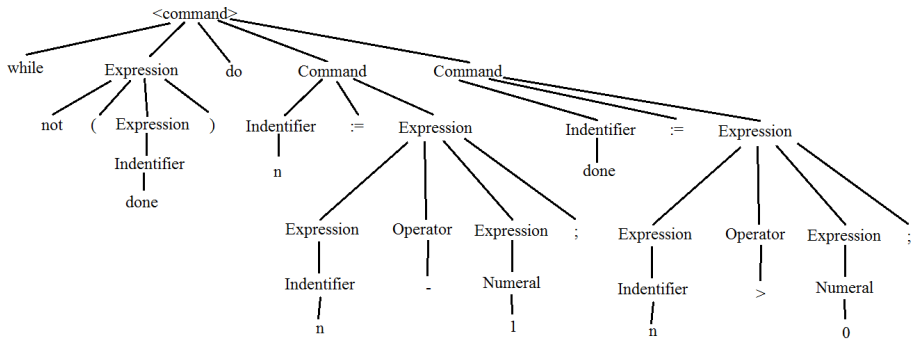


Answer to question ten on page sixteen

The grammar is ambiguous because there are two different trees for "a".
We can write an unambiguous grammar for the grammar as next grammar.
<First Expression> ::= <Second Expression> | <Second Expression> +
<First Expression>
<Second Expression> ::= <Third Expression> | <Third Expression> *
<Second Expression>
<Third Expression> ::= (<First Expression>) | a | b | c

Answer to question two on page twenty nine

We can write the abstract syntax tree as next syntax tree.



Answer to question one on page seventy one

The attribute grammar is following statement.

<string literal>::= <numeral> H <string>

Condition: Value (<numeral>) = length (<string>)

<numeral>::= < $numeral_2$ > <digit>

Value (<numeral>) ← Value (< $numeral_2$ >)* 10+ Value (<digit>)

<numeral>::= <digit>

Value (<numeral>) ← Value (<digit>)

<string>::= < $string_2$ > <character>

length (<string>) ← length (< $string_2$ >)+ 1

<string>::= <character>

length (<string>) ← 1

<string>::= ϵ

length (<string>) ← 0

Answer to question four on page seventy one

The bellow attribute grammar satisfies both conditions.

$< roman >::=< hundreds >< tens >< units >$

$value(< roman >) \leftarrow value(< hundreds >) + value(< tens >) + value(< units >)$

$< hundreds >::=< low\ hundreds >$

Condition: $length(< low\ hundreds >) < 4$

$value(< hundreds >) \leftarrow value(< low\ hundreds >)$

$< hundreds >::= CD$

$value(< hundreds >) \leftarrow 400$

$< hundreds >::= D < low\ hundreds >$

Condition: $length(< low\ hundreds >) < 4$

$value(< hundreds >) \leftarrow value(< low\ hundreds >) + 500$

$< hundreds >::= CM$

$value(< hundreds >) \leftarrow 900$

$< low\ hundreds >::= \epsilon$

$length(< low\ hundreds >) \leftarrow 0$

$value(< low\ hundreds >) \leftarrow 0$

$< low\ hundreds >::=< low\ hundreds_2 > C$

$length(< low\ hundreds >) \leftarrow length(< low\ hundreds_2 >) + 1$

$value(< low\ hundreds >) \leftarrow value(< low\ hundreds_2 >) + 100$

$< tens >::=< low\ tens >$

Condition: $length(< low\ tens >) < 4$

$value(< tens >) \leftarrow value(< low\ tens >)$

$< tens >::= XL$

$value(< tens >) \leftarrow 40$

$< tens >::= L < low\ tens >$

Condition: $length(< low\ tens >) < 4$

$value(< tens >) \leftarrow value(< low\ tens >)$

$< tens >::= XC$

$value(< tens >) \leftarrow 90$

$< low\ tens >::= \epsilon$

$length(< low\ tens >) \leftarrow 0$

$value(< low\ tens >) \leftarrow 0$

$< low\ tens >::=< low\ tens_2 > X$

$length(< low\ tens >) \leftarrow length(< low\ tens_2 >) + 1$

$value(< low\ tens >) \leftarrow value(< low\ tens_2 >) + 10$

$< units >::=< low\ units >$

Condition: $length(< low\ units >) < 4$

$value(< units >) \leftarrow value(< low\ units >)$

$< units >::= IV$

$value(< low\ units >) \leftarrow 4$

$< units >::= V < low\ units >$

Condition: $length(< low\ units >) < 4$

$value(< units >) \leftarrow value(< low\ units >) + 5$

$< units >::= IX$

$value(< units >) \leftarrow 9$

$< low\ units >::= \epsilon$

$length(< low\ units >) \leftarrow 0$

$value(< low\ units >) \leftarrow 0$

$< low\ units >::=< low\ units_2 > I$

$length(< low\ units >) \leftarrow length(< low\ units_2 >) + 1$

$value(< low\ units >) \leftarrow value(< low\ units_2 >) + 1$

Answer to question five on page seventy one

We can modify first version as following grammar.

$< binary\ numeral >::=< binary\ digits >_1 . < binary\ digits >_2$

$Val(< binary\ numeral >) \Leftarrow Val(< binary\ digits >_1) + Val(< binary\ digits >_2)/2^{Len(<binary\ digits>_2)}$

$< binary\ numeral >::=< binary\ digits >_1$

$Val(< binary\ numeral >) \Leftarrow Val(< binary\ digits >_1)$

$< binary\ numeral >::=< binary\ digits >_1 .$

$Val(< binary\ numeral >) \Leftarrow Val(< binary\ digits >_1)$

$< binary\ numeral >::= . < binary\ digits >_2$

$Val(< binary\ numeral >) \Leftarrow Val(< binary\ digits >_2)/2^{Len(<binary\ digits>_2)}$

$< binary\ digits >::=< binary\ digits >_2 < bit >$

$Val(< binary\ digits >) \Leftarrow 2 * Val(< binary\ digits >_2) + Val(< bit >)$

$Len(< binary\ digits >) \Leftarrow Len(< binary\ digits >_2) + 1$

$< binary\ digits >::=< bit >$

$Val(< binary\ digits >) \Leftarrow Val(< bit >)$

$Len(< binary\ digits >) \Leftarrow 1$

$< bit >::= 0$

$Val(< bit >) \Leftarrow 0$

$< bit >::= 1$

$Val(< bit >) \Leftarrow 1$

We can modify second version as following grammar.

$< binary\ numeral >::=< binary\ digits > . < fraction\ digits >$

$Val(< binary\ numeral >) \leftarrow Val(< binary\ digits >) + Val(< fraction\ digits >)$

$Pos(< binary\ digits >) \leftarrow 0$

$< binary\ numeral >::=< binary\ digits >$

$Val(< binary\ numeral >) \leftarrow Val(< binary\ digits >)$

$Pos(< binary\ digits >) \leftarrow 0$

$< binarynumeral >::=< binary\ digits > .$

$Val(< binary\ numeral >) \leftarrow Val(< binary\ digits >)$

$Pos(< binary\ digits >) \leftarrow 0$

$< binary\ numeral >::= . < fraction\ digits >$

$Val(< binary\ numeral >) \leftarrow Val(< fraction\ digits >)$

$< binary\ digits >::=< binary\ digits >_2< bit >$

$Val(< binary\ digits >) \leftarrow Val(< binary\ digits >_2) + Val(< bit >)$

$Pos(< binary\ digits >_2) \leftarrow Pos(< binary\ digits >) + 1$

$Pos(< bit >) \leftarrow Pos(< binary\ digits >)$

$< binary\ digits >::=< bit >$

$Val(< binary\ digits >) \leftarrow Val(< bit >)$

$Pos(< bit >) \leftarrow Pos(< binary\ digits >)$

$< fraction\ digits >::=< fraction\ digits >_2< bit >$

$Val(< fraction\ digits >) \leftarrow Val(< fraction\ digits >_2) + Val(< bit >)$

$Len(< fraction\ digits >) \leftarrow Len(< fraction\ digits >_2) + 1$

$Pos(< bit >) \leftarrow -Len(< fraction\ digits >)$

$< fraction\ digits >::=< bit >$

$Val(< fraction\ digits >) \leftarrow Val(< bit >)$

$Len(< fraction\ digits >) \leftarrow 1$

$Pos(< bit >) \leftarrow -1$

$< bit >::= 0$

$Val(< bit >) \leftarrow 0$

$< bit >::= 1$

$Val(< bit >) \leftarrow 2^{Pos(<bit>)}$

Answer to question ten on page seventy one

We can write the attribute grammar as next attribute.
$< Initiate >::=< expr >$
Condition : $OperatorSize(< expr >) \leq 8$
Condition : $MultiplicationSize(< expr >) \geq DivisionSize(< expr >)$ *or* $SubtractionSize(< expr >) = 0$
Condition : $ParenthesisSize(< expr >) \leq 3$
$< expr >::=< expr >_2 + < T1 >$
$ParenthesisSize(< expr >) \leftarrow Maximum(ParenthesisSize(< expr >_2), ParenthesisSize(< T1 >))$
$OperatorSize(< expr >) \leftarrow OperatorSize(< expr >_2) + OperatorSize(< T1 >) + 1$
$MultiplicationSize(< expr >) \leftarrow MultiplicationSize(< expr >_2) + MultiplicationSize(< T1 >)$
$DivisionSize(< expr >) \leftarrow DivisionSize(< expr >_2) + DivisionSize(< T1 >)$
$SubtractionSize(< expr >) \leftarrow SubtractionSize(< expr >_2) + SubtractionSize(< T1 >)$
$< expr >::=< expr >_2 - < T1 >$
$ParenthesisSize(< expr >) \leftarrow Maximum(ParenthesisSize(< expr >_2), ParenthesisSize(< T1 >))$
$OperatorSize(< expr >) \leftarrow OperatorSize(< expr >_2) + OperatorSize(< T1 >) + 1$
$MultiplicationSize(< expr >) \leftarrow MultiplicationSize(< expr >_2) + MultiplicationSize(< T1 >)$
$DivisionSize(< expr >) \leftarrow DivisionSize(< expr >_2) + DivisionSize(< T1 >)$
$SubtractionSize(< expr >) \leftarrow SubtractionSize(< expr >_2) + SubtractionSize(< T1 >) + 1$
$< expr >::=< T1 >$
$ParenthesisSize(< expr >) \leftarrow ParenthesisSize(< T1 >)$
$OperatorSize(< expr >) \leftarrow OperatorSize(< T1 >)$
$MultiplicationSize(< expr >) \leftarrow MultiplicationSize(< T1 >)$
$DivisionSize(< expr >) \leftarrow DivisionSize(< T1 >)$
$SubtractionSize(< expr >) \leftarrow SubtractionSize(< T1 >)$
$< T1 >::=< T1 >_2 * < T2 >$
$ParenthesisSize(< T1 >) \leftarrow Maximum(ParenthesisSize(< T1 >_2$

$), ParenthesisSize(< T2 >))$

$OperatorSize(< T1 >) \leftarrow OperatorSize(< T1 >_2$
$) + OperatorSize(< T2 >) + 1$

$MultiplicationSize(< T1 >) \leftarrow MultiplicationSize(< T1 >_2$
$) + MultiplicationSize(< T2 >) + 1$

$DivisionSize(< T1 >) \leftarrow DivisionSize(< T1 >_2$
$) + DivisionSize(< T2 >)$

$SubtractionSize(< T1 >) \leftarrow SubtractionSize(< T1 >_2$
$) + SubtractionSize(< T2 >)$

$< T1 >::=< T1 >_2 \ / < T2 >$

$ParenthesisSize(< T1 >) \leftarrow Maximum(ParenthesisSize(< T1 >_2$
$), ParenthesisSize(< T2 >))$

$OperatorSize(< T1 >) \leftarrow OperatorSize(< T1 >_2$
$) + OperatorSize(< T2 >) + 1$

$MultiplicationSize(< T1 >) \leftarrow MultiplicationSize(< T1 >_2$
$) + MultiplicationSize(< T2 >)$

$DivisionSize(< T1 >) \leftarrow DivisionSize(< T1 >_2$
$) + DivisionSize(< T2 >) + 1$

$SubtractionSize(< T1 >) \leftarrow SubtractionSize(< T1 >_2$
$) + SubtractionSize(< T2 >)$

$< T1 >::=< T2 >$

$ParenthesisSize(< T1 >) \leftarrow ParenthesisSize(< T2 >)$

$OperatorSize(< T1 >) \leftarrow OperatorSize(< T2 >)$

$MultiplicationSize(< T1 >) \leftarrow MultiplicationSize(< T2 >)$

$DivisionSize(< T1 >) \leftarrow DivisionSize(< T2 >)$

$SubtractionSize(< T1 >) \leftarrow SubtractionSize(< T2 >)$

$< T2 >::=< T3 >\uparrow < T2 >_2$

$ParenthesisSize(< T2 >) \leftarrow Maximum(ParenthesisSize(< T3 >$
$), ParenthesisSize(< T2 >_2))$

$OperatorSize(< T2 >) \leftarrow OperatorSize(< T3 >$
$) + OperatorSize(< T2 >_2) + 1$

$MultiplicationSize(< T2 >) \leftarrow MultiplicationSize(< T3 >$
$) + MultiplicationSize(< T2 >_2)$

$DivisionSize(< T2 >) \leftarrow DivisionSize(< T3 >) + DivisionSize(<$
$T2 >_2)$

$SubtractionSize(< T2 >) \leftarrow SubtractionSize(< T3 >$
$) + SubtractionSize(< T2 >_2)$

$< T2 >::=< T3 >$

$ParenthesisSize(< T2 >) \leftarrow ParenthesisSize(< T3 >)$
$OperatorSize(< T2 >) \leftarrow OperatorSize(< T3 >)$
$MultiplicationSize(< T2 >) \leftarrow MultiplicationSize(< T3 >)$
$DivisionSize(< T2 >) \leftarrow DivisionSize(< T3 >)$
$SubtractionSize(< T2 >) \leftarrow SubtractionSize(< T3 >)$
$< T3 >::= (< expr >)$
$ParenthesisSize(< T3 >) \leftarrow ParenthesisSize(< expr >) + 1$
$OperatorSize(< T3 >) \leftarrow OperatorSize(< expr >)$
$MultiplicationSize(< T3 >) \leftarrow MultiplicationSize(< expr >)$
$DivisionSize(< T3 >) \leftarrow DivisionSize(< expr >)$
$SubtractionSize(< T3 >) \leftarrow SubtractionSize(< expr >)$
$< T3 >::= a|b|c$
$ParenthesisSize(< T3 >) \leftarrow 0$
$OperatorSize(< T3 >) \leftarrow 0$
$MultiplicationSize(< T3 >) \leftarrow 0$
$DivisionSize(< T3 >) \leftarrow 0$
$SubtractionSize(< T3 >) \leftarrow 0$

Answer to question eleven on page seventy one

Answer to part a

The following attribute grammar satisfies the requirements for this part.
$< binary\ tree >::= nil$
$Height(< binary\ tree >) \Leftarrow 0$
$< binary\ tree >::= (< binary\ tree >_2 < value > < binary\ tree >_3)$
Condition : $|Height(< binary\ tree >_2) - Height(< binary\ tree >_3)| < 1$
$Height(< binary\ tree >) \Leftarrow 1 + Maximum(Height(< binary\ tree >_2), Height(< binary\ tree >_3))$
$< value >::= < digit >$
$< value >::= < value >_2 < digit >$
$< digit >::= 0|1|2|3|4|5|6|7|8|9$

Answer to part b

The following attribute grammar satisfies the requirements for this part.

$< binary\ tree >::= nil$

$UpperBound(< binary\ tree >) \Leftarrow -\infty$

$LowerBound(< binary\ tree) \Leftarrow +\infty$

$< binary\ tree >::= (< binary\ tree >_2 < value > < binary\ tree >_3)$

$UpperBound(< binary\ tree >) \Leftarrow Maximum(UpperBound(< binary\ tree >_3), Value(< value >))$

$LowerBound(< binary\ tree) \Leftarrow Minimum(LowerBound(< binary\ tree >_2), Value(< value >))$

Condition : $LowerBound(< binary\ tree >) < Value(< value >) \leq UpperBound(< binary\ tree >)$

$< value >::=< digit >$

$Value(< value >) \Leftarrow Value(< digit >)$

$< value >::=< value >_2 < digit >$

$Value(< value >) \Leftarrow Value(< value >_2) * 10 + Value(< digit >)$

$< digit >::= 0|1|2|3|4|5|6|7|8|9$