Solutions to Selected Exercises for Assignment 2

Problem 1 *(20 points)*
Prove that a language is also Turing complete if it (just) contains the following constructs:

boolean expressions
integer variables
integer arithmetic
sequential execution of statements
assignment statements
while-loops

Note that "proof by specific example" is not a valid technique here. (Hint: Language equivalence). Note that the "break" command is not part of any of these languages.

Solution. Intuition/insight: The approach is to show that the proposed language can simulate all of the constructs of either language L1 or language L2. Language L1 is a closer match so we choose L1. The only difference between L1 and the proposed language are *if-statements*. Thus, we try to show that while-loops can be used to simulate if-statements.

Attempt 1. The following is an attempt to show "by example" that while-loops can be used to simulate the behavior of if-then statements.

if x = 1 then y = 2
≡
while x= 1 do { x := x + 1; y = 2; }

Such a solution is not general and would not receive high marks.

Attempt 2. Here is a more general solution.

if <BE> then <stmt>
≡
x := true
while <BE> and x do { <stmt>; x := false }

where <BE> stands for any boolean expression, and <stmt> stands for any statement.

In the above solution, the assumption is that "x" is a unique identifier. Without this assumption the above translation would be incorrect in general. Furthermore, this assumption should be stated explicitly as part of the solution. And finally, this solution is only correct if the evaluation of the boolean expression, <BE>, does NOT cause side-effects. Note that the while loop will cause <BE> to be evaluated twice in contrast to the if-then statement which will cause <BE> to be evaluated only once.

Attempt 3. Here is the correct solution.

if <BE> then <stmt>
≡
x := <BE>; **where** x is a unique identifier.
while x do { <stmt>; x := false }

Problem 2 *(10 points)*

Give a BNF grammar that approximates the syntactic structure of a textbook. The description should be quite detailed but need not be exact. An example of an English approximation of the structure of a book might begin as follows: A book has a cover page containing the title. After the cover page, there is a table of contents containing chapter titles as well as starting page numbers. Then comes a preface which consists of several paragraphs of text. Note that the preface is part of the contents. Chapters consist of sections that contain one or more paragraphs. Each chapter must end with an exercises section. Paragraphs consist of one or more sentences, etc. etc.

After constructing your grammar, describe five areas where the grammar fails to precisely capture the structure of the book. For example, if your description allows a book to contain an arbitrary number of chapters, then the grammar will not be able to enforce that chapters are numbered consecutively. For example, the chapter sequence 1, 3, 2 would be considered to be syntactically legal according to the grammar.

Solution The grammar below describes a textbook that was once used in this class. In the BNF below, nonterminal symbols begin with a capital letter, terminal symbols denoting basic tokens (e.g., keywords) are enclosed in quotes, and terminal symbols denoting sets of tokens (e.g., identifier, integer, etc.) are in ***italics***.

Important: Notice that this BNF is not complete in the sense that not all nonterminal symbols are defined; that is, they occur in a production having "dot dot dot" as its right-hand side. **However**, all nonterminals shown can be reached from the nonterminal Book which is the start symbol of this grammar. This kind of "reachability" is something that I will look for when grading.

Book	::=	Front_Cover
		Boiler_Plate
		Table_of_Contents
		Preface_Section
		Chapter_list
		Appendix_list
		Bibliography_Section
		Index_Section
		Back_cover

Front_cover	::=	Title Authors
Title	::=	“Programming” “Languages” “Principles” “and” “Paradigms”
Authors	::=	“Allen” “Tucker” “Robert” “Noonan”
Boiler_Plate	::=	Word_list
Table_of_Contents	::=	Contents Chapter_info
Chapter_info	::=	Number Chapter_title Number
Number	::=	***integer*** “.” ***integer*** | ***integer***
Preface_Section	::=	Preface Text
Chapter_list	::=	Chapter Chapter_list | Chapter
Chapter	::=	Chapter_title Number Word_list Chapter_outline Text Exercise_Section
Chapter_title	::=	Word_list
Text	::=	Paragraph_list
Paragraph_list	::=	Paragraph Paragraph_list | Paragraph
Paragraph	::=	Marker Sentence_list | Diagram | Figure | Table
Marker	::=	Number | ***letter*** “.” Number
Sentence_list	::=	Sentence Sentence_list | Sentence
Sentence	::=	Word_list “.”
Word_list	::=	***word*** Word_list | ***word***
Exercise_Section	::=	“EXERCISES” Problem_list
Problem_list	::=	Problem Problem_list | Problem
Problem	::=	Number Text
Appendix_list	::=	Appendix Appendix_list | Appendix
Appendix	::=	Chapter_title ***letter*** Word_list Text
Bibliography_Section	::=	Word_list
Index_Section	::=	...
Back_cover	::=	...
Contents	::=	...
Preface	::=	...
Chapter_outline	::=	...
Diagram	::=	...
Figure	::=	...
Table	::=	...

Analysis of grammar. The above grammar fails to accurately describe the structure of the book in many many places.

1. The bibliography section is not simply a word list.
2. English sentences are more than simply word lists. That is, not all word lists correspond to English sentences.
3. The boiler plate section should include precise information such as the title of the book, publisher, ISBN number, etc.
4. Chapters should be numbered in consecutive order.
5. Appendices should be lettered in consecutive order.
6. Problems should be numbered in consecutive order.
7. Subsections of paragraphs should begin with the number of the paragraph in which they occur.

Problem 6 *(10 points)*

You are given the following alphabet: { *a, b, c, d* }.

1. *Write a regular expression describing the set of all identifiers (i.e., strings) having at most 2 occurrences of the letter c. For example, the identifier "abbcdaac" would belong to this set, but the identifier "aabcdaacc" would not.*

   (a) *Solution: (a+b+d)(a+b+d)* + (a+b+d)*c(a+b+d)* + (a+b+d)*c(a+b+d)*c(a+b+d)**

2. *Write a regular expression describing the set of all identifiers having at least 2 occurrences of the letter c.*

   (a) *Solution: (a+b+c+d)*c(a+b+c+d)*c(a+b+c+d)**

Problem 7 *(15 points)*

Consider two definitions of the language of mathematical expressions. This language contains the following operators and constants:

- *Arithmetic Operators and their signatures.*

Name of Operator	*Operator Symbol*		*Type Expression*
addition	+	:	*integer * integer → integer*
subtraction	−	:	*integer * integer → integer*
multiplication	∗	:	*integer * integer → integer*
division	/	:	*integer * integer → integer*
exponentiation	∗∗	:	*integer * integer → integer*

Note that the signature of an operator is an expression of the form $f : \tau$ where f is the symbol denoting the operator and τ is a type expression that describes the types of the operands of f and the type of its result. For example, integer addition is a binary operator that takes two integers as its operands and produces an integer as its result. Formally, we will write the signature of addition as follows:

$$+ : integer * integer \rightarrow integer$$

In the expression $integer * integer$ the $*$ denotes a domain cross-product. In particular, in this context $*$ does ***NOT*** *denote multiplication.*

- *Constants*
  - $0, 1, 2, 3, ...$

Grammar 1

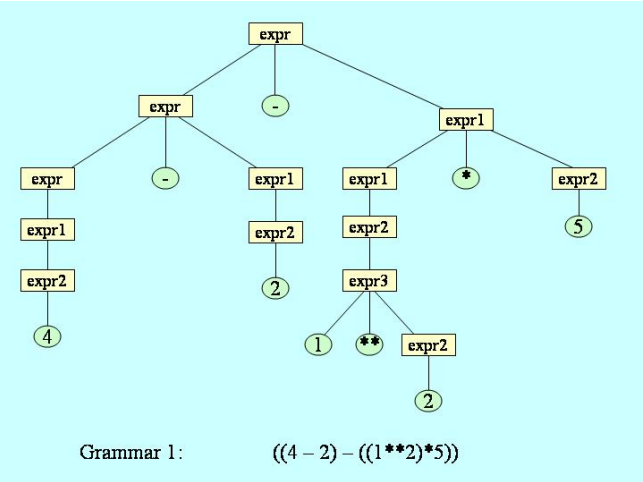
expr	*::=*	*expr1* | *expr + expr1* | *expr - expr1*
expr1	*::=*	*expr2* | *expr1 * expr2* | *expr1 / expr2*
expr2	*::=*	*integer* | *- integer* | *expr3*
expr3	*::=*	*integer ** expr2*

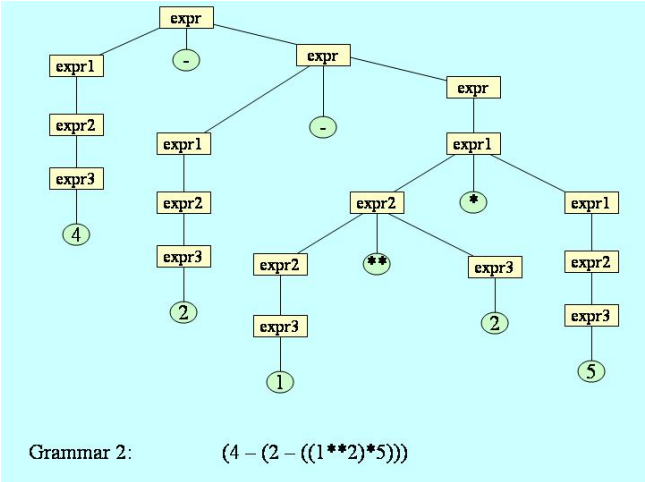
Grammar 2

expr	*::=*	*expr1* | *expr1 + expr* | *expr1 - expr*
expr1	*::=*	*expr2* | *expr2 * expr1* | *expr2 / expr1*
expr2	*::=*	*expr2 ** expr3* | *expr3*
expr3	*::=*	*integer* | *- integer*

1. *Using both grammars, draw a parse tree for the expression:* $4 - 2 - 1 * *2 * 5$
2. *What is the difference between these two grammars?*
3. *Describe the practical significance/impact of this difference and give arguments in favor of choosing one grammar over the other.*

Solution to part 1.





Solution to part 2. These grammars differ in the precedence and associativity of their arithmetic operators. Additionally, grammar 1 cannot be used to generate expressions like: $5 + \ -1^2$ however an expression like $5 - 1^2$ can be generated.

- Grammar 1:
  - Left associative operators: $\{+, -, *, /\}$
  - Precedence: $\{+, -\} < \{*, /\} <$ unary minus $< \{**\}$
  - The exponentiation operator is written in a manner that is consistent with right-associativity.
- Grammar 2:
  - Right associative operators: $\{+, -, *, /\}$
  - Left associative operators: $\{**\}$
  - Precedence: $\{+, -\} < \{*, /\} < \{**\} <$ unary minus

Solution to part 3. Grammar 1 supports a syntax whose syntax directed semantics is consistent with a reasonable mathematical interpretation of the expressions in the language. Grammar 2 incorrectly has the binary minus operator as being right associative when in fact it is left associative. Grammar 2 also has exponentiation as being left associative which is incorrect.