Spring 2017 CSE Qualifying Exam
CSCE 531, Compilers

1. **LR-Parsing.**

   (a) Give definitions of FIRST(α) and FOLLOW(X).

   (b) Consider the following augmented grammar G with start symbol S':

$$\begin{aligned} S' &\rightarrow S \\ S &\rightarrow S\ S\ + \\ S &\rightarrow S\ S\ * \\ S &\rightarrow a \end{aligned}$$

   For the grammar G above generate all of the LR(1) sets of items along with complete transition information for a canonical LR(1) parser

   (c) Using the partial Sets-of-Items constructed in part (b) construct the action table.

   (d) Describe in detail how the LR Parsing Algorithm will proceed in general when the state is s, the next token is t and the stack contents is $X_0, X_1, \ldots, X_{top-1}, X_{top}$.

2. **Syntax-Directed Definitions.** Consider the following standard LR (bottom-up) grammar for boolean expressions with constants TRUE (1) and FALSE (0), Boolean variables (v), operations AND ($\wedge$), OR ($\vee$), and NOT ($\neg$), and parentheses (S is the start symbol):

$$
\begin{array}{lcl}
S & ::= & E \\
E & ::= & E_1 \vee T \\
E & ::= & T \\
T & ::= & T_1 \wedge F \\
T & ::= & F \\
F & ::= & \neg F_1 \\
F & ::= & 0 \\
F & ::= & 1 \\
F & ::= & v \\
F & ::= & (E)
\end{array}
$$

(Subscripts have been added to distinguish different occurrences of the same nonterminal in the same production.) The Boolean constants and operators have the usual semantics, with AND having higher precedence than OR and lower precedence than NOT.

Add semantic rules to the grammar above that, given an input expression, produce a simplified equivalent expression. The only two rules you are to use to simplify an expression are (1) those that short-circuit the evaluation of AND, OR, and NOT, and (2) those that notice AND or OR applied to *syntactically* identical operands (after they have been simplified). You should also omit surrounding parentheses if the simplified expression inside is a constant, e.g., (0) simplifies to 0. Do not remove parentheses in any other situations, however. See more examples below.

Your resulting expression should be passed as a string attribute to $S.output$. Assume that the terminal v has a *text* attribute that contains the string representing the variable (identifier). This attribute does *not* tell you the truth value of that variable, so you cannot make any assumptions about its truth value. You may use '+' in your actions to denote string concatenation, and you may use == to test for string equality; please surround string constants with double quotes.

Examples:

Input	*S.output*	**Comment**
$x \wedge (y \vee z)$	$x \wedge (y \vee z)$	no simplification
$\neg y \vee y$	$\neg y \vee y$	no simplification!
$(x \wedge y) \vee (x \wedge y)$	$(x \wedge y)$	syntactically identical operands
$x \wedge y \vee x \wedge y$	$x \wedge y$	syntactically identical operands
$(x \wedge y) \vee x \wedge y$	$(x \wedge y) \vee x \wedge y$	no simplification!
$\neg\neg x$	$\neg\neg x$	no simplification!
$0 \vee x$	x	0 disappears in a disjunction
$y \wedge 0$	0	part of the conjunction is false
$1 \wedge \neg y$	$\neg y$	1 disappears in a conjunction
$x \vee x \wedge x$	x	simplified operands syntactically identical
$x \wedge 1 \vee x$	x	simplified operands syntactically identical
$x \wedge \neg 1$	0	two simplifications in a row
(x)	(x)	no simplification!
$x \vee (1)$	1	simplification safely removes parens here

3. **Control Flow and Liveness Analysis.** Consider the intermediate code below.

```
 1      prod = 1
 2      i = 0
 3  L0: j = 1
 4  L1: j = j + 1
 5      if j > n goto L3
 6      t1 = i + j
 7      prod = prod * t1
 8      t2 = prod
 9      a[t1] = i
10      goto L1
11  L2: i = t2 + 1
12  L3: i = i + 1
13      if i <= n goto L0
14      if prod < i goto L4
15      prod = prod - i
16      i = i - 1
17      goto L1
18  L4: no-op
19      return
```

Assume that there are no entry points into the code from outside other than at the start.

(a) (20% credit) Decompose the code into basic blocks B1,B2, ..., giving a range of line numbers for each.

(b) (20% credit) Draw the control flow graph, and describe any unreachable code.

(c) (40% credit) Fill in a 19-row table listing which variables are live at which control points. Treat `a` as a single variable. Assume that `n` and `prod` are the only live variables immediately before line 19 (the only exit point). Your table should look like this:

Before line	**Live variables**
1	...
2	...
3	...
...	...
19	...

(d) (20% credit) Describe any simplifying transformations that can be performed on the code (i.e., transformations that preserve the semantics but reduce (i) the complexity of an instruction, (ii) the number of instructions, (iii) the number of branches, or (iv) the number of variables).

1. AMAT: In the system we are analyzing the memory has:
   - Separate L1 instruction and data caches, HitTime = Processor Cycle Time
   - 64KB L1 instruction cache with 1% miss rate, 32B blocks
   - 64KB L1 data cache with 10% miss rate, 32B blocks
   - 512K L2 unified cache with 64B blocks, local miss rate 50%, Hit Time = 6 cycles,
   - Main Memory Access time is 100 cycles for the first 128 bits and subsequent 128 bit chunks are available every 8 cycles.
   - Both L1 caches are direct mapped, L2 four-way associative.
   - Assume there are no misses to main memory.

   (a) What is the Miss Penalty for accesses to L2?

   (b) What is the average memory access time for instruction references?

   (c) What is the average memory access time for data references?

   (d) Assume the only memory reference instructions are loads(25%) and stores(5%). What percentage of total memory references are data references?

   (e) What is the Average memory access time?

2. The Niagara (T1) processor was a hyperthreaded design with 8 logical cores.

   (a) What type of system was the Niagara(=T1 processor) designed to support?

   (b) What design decisions (e.g., what hardware was included, what was not included) were made to address this goal?

   (c) What changes to the classical 5-stage pipeline are there in the T1? (include additional hardware and discuss operation)

   (d) How did the Niagara system handle stalls due to mis-predicted branches?

   (e) The T5, a fifth generation of this processor, had 16 cores each supporting 8 threads for a total of 128 threads. Assuming ideal situation, i.e., no stalls, no idle processors what is the best CPI?

3. ROB - single issue

   (a) Explain the contents of the ROB.

   (b) What are the $Q'_j s$ in Tomasulo's and what are they in ROB?

   (c) Explain what happens when a branch instruction gets to the front of the ROB queue.

   (d) What control hardware is necessary for loading the value field of a reservation station?

   (e) Trace - Assume the following:

   - **Ten ROB slots.**
   - 1 integer Execution unit, 5 reservation stations, one cycle to execute.
   - 1 Floating Add Unit, 3 reservation stations, 10 cycles to execute.
   - 1 Floating MULT Unit, 2 reservation stations, 15 cycles to execute.
   - 1 Load unit with only two reservation stations.
   - Loads on hits require 2 clock cycles, both performed in the Load Unit.
   - Functional units are not pipelined!!!!
   - There is no forwarding between functional units; results are communicated by the common data bus (CDB).
   - The execution stage (EX) does both the effective address calculation and the memory access for loads and stores.
   - Assume all memory references are hits and branch predictions are branch taken and are correct.
   - The issue (IS) and write-back (WB) result stages each require one clock cycle.
   - Assume that the Branch on Not Equal to Zero (BNEZ) instruction requires one clock cycle.

```
loop:  LD     F0, 0(R1)
       LD     F2, 1000(R1)
       MUL.D  F4, F2, F0
       MUL.D  F8, F8, F4
       DADDIU R1, R1, +8
       BNE    R1, R2, loop
```

Fill in the table below for as much as you can assuming the BNE is taken and succeeds. (Do not add rows to the table.)

Iter.	Instruction	Issues at	Execute/memory	Write CDB	Commit	Comment

Algorithms

1. Let some positive-valued function $T(n)$ satisfy the following recurrence for all sufficiently large n:

$$T(n) = \sqrt{n}T(\sqrt{n}) + n$$

Use the substitution method to **prove** that $T(n) = \Theta(n \lg \lg n)$. You are only required to show the inductive step; you need not show how base cases are handled. You may assume that any implicit floors or ceilings are of no consequence.

2. **(Carpenter's Ruler Problem)** You are given a rod N centimeters long, divided into segments of lengths $a_1, a_2, \ldots, a_n$ centimeters (each a_i is a positive integer). There is a crease between every pair of adjacent segments allowing you (if you choose) to fold the rod 180° at the crease. You want to choose creases to fold so that the folded rod is as short as possible. (Assume the thickness of the rod is negligible.)

Given a list of $\langle a_1, \ldots, a_n \rangle$ segment lengths (positive integers) and an integer L, describe an algorithm that decides (yes or no) whether the rod can be folded into something of length $\leq L$. Your algorithm should run in time $O(N^4)$, where $N = \sum_{i=1}^{n} a_i$ is the total length of the rod.

Here is an equivalent formulation of the problem: Given a list $A = \langle a_1, \ldots, a_n \rangle$ of positive integers, define a *fold* of A to be a list $\langle d_1, \ldots, d_n \rangle$ where each $d_i \in \{-1, 1\}$. The *max* M of a fold is

$$M = \max_{0 \leq k \leq n} \sum_{i=1}^{k} d_i a_i \ .$$

Similarly, the *min* of the fold is

$$m = \min_{0 \leq k \leq n} \sum_{i=1}^{k} d_i a_i \ .$$

(Imagine the folded rod lying along the real line with one end anchored at the origin. Then $M \geq 0$ is the extent of the folded rod in the positive direction, and $m \leq 0$ is the extent of the folded rod in the negative direction.) Given a list A as above and an integer L, you want to determine whether there is a fold of A such that $M - m \leq L$.

Hint: Use dynamic programming, filling in the entries of a 4-dimensional Boolean array

$$T[0 \ldots n,\ (-N) \ldots 0,\ 0 \ldots N,\ (-N) \ldots N] \ ,$$

Where $T[k, \ell, h, s]$ is true iff there is a fold $\langle d_1, \ldots, d_k \rangle$ of $\langle a_1, \ldots, a_k \rangle$ (the first k segments of the rod) with min $\geq \ell$ and max $\leq h$ and satisfying $\sum_{i=1}^{k} d_i a_i = s$. Fill in T in order of increasing k, then use T to determine your final answer.

3. Let $G = (V, E)$ be a directed graph with edge weight function $w : E \to [1, \infty)$ giving each edge $e \in E$ a real-number weight ≥ 1. Define the **multiplicative cost** of a path through this graph to be the product of the weights of the edges along the path. Let $s, t \in V$ be a pair of vertices in this graph. **Describe** an algorithm that, given G, s, and t, finds the path in G from s to t with the smallest multiplicative cost. **Explain** why your algorithm works, in enough detail to convince an intelligent but skeptical reader that it is correct. You may assume that scalar functions and operations on real numbers take constant time, regardless of how big the numbers are.