Jay Bolton Week 1

Logic for Applications

ch1: 1, 2, 4, 6, 7
ch2: 1, 2, 3, 5
ch3: 1, 2, 3

Ch 1.
1.
```
     0
    / \
   1   2
  /
 3
```

2.
```
     0
    / \
   1   2
      /|\...
(node 2 has infinite immediate successors)
```

4.
In definition 1.2, it is stated that every node is well ordered by $<_T$. By the definition of well ordering, there is a least element of every subset. Suppose that node A has two immediate predecessors X and Y. Therefore, both X and Y are less than A and X !< Y and Y !< X, but by hypothesis X != Y, or they would be the same node. Now let us choose the subset of {X,Y}, which has no least element. This contradicts the definition of well ordering. Thus, A can only have one immediate predecessor.

6.
By the definition of well ordering, every subset of S must have a least element. It follows that the subset consisting of the entire set S must have a least element. This would be impossible in an infinite descending chain which has no least element.

7.
Number pairs are transitive, irreflexive, and obey the trichotomy law. This is apparent from the definition of lexicographic ordering: for the two pairs [a,b] and [c,d], a and c are well ordered natural numbers, and on the occasion that they are equal, we compare b and d, which are also well ordered natural numbers. The subset of all pairs of natural numbers also has a least element – [0.0] -- and for each proper subset, there is a least element, because we are always comparing natural numbers.

Ch 2.
1.
a, d, f

2.
a.

Basis: {A, B, C}
Inductive Steps:
(A v B)
(~(A v B))
((~(A v B)) ^ C)

b.
Basis: {A, B, C}
Inductive steps:
(A ^ B)
((A ^ B) v C)
Error: we did not surround the expression with parenthesis where the OR was added.

f.
{A, B, C, D}
(C v B)
((C v B) ^ A)
(((C v B) ^ A) ↔ D)

3.
n_l is number of left parens and n_r is number of right parens
Basis:
E_0 = {A, B}
$n_l = n_r = 0$
IH: $n_l = n_r$ for up to k applications of the recursive step.
Induction:
by the IH, α and β each have $n_l = n_r$. Our options in the recursive step are (α ^ β), (α v β), (~α), (~β), (α → β), or (α ↔β). Each of these possibilities add exactly two parentheses. Let $n_l(\alpha) + n_l(\beta) = x$, and $n_r(\alpha) + n_r(\beta) = y$. (α ^ β) has $x + y + 1$ left parens and $x + y + 1$ right parens, which are equal. Thus they will always be equal. It is the same procedure for the other recursive steps, which all similarly add 2 parentheses.

5.
a.
(A → B) → C
(~A v B) → C
~(~A v B) v C
(A ^ ~B) v C
b.
(A ↔ B) v (~C)
((~A ^ ~B) v (A ^ B)) v (~C)
(~A ^ ~B) v (A ^ B) v (~C)

Ch 3.
1.
(A v (~A))

Jay Bolton Week 1

V_1
A = True
(T v (~T)) = T

V_2
A = False
(F v (~F)) = T

V_1
A = True
B = True
$(((A \rightarrow B) \rightarrow A) \rightarrow A)$
$(((T \rightarrow T) \rightarrow T) \rightarrow T)$
$(((T) \rightarrow T) \rightarrow T)$
$((T) \rightarrow T)$
T

V_2
A = True
B = False
$(((A \rightarrow B) \rightarrow A) \rightarrow A)$
$(((T \rightarrow F) \rightarrow T) \rightarrow T)$
$(((F) \rightarrow T) \rightarrow T)$
$((T) \rightarrow T)$
T

V_3
A = False
B = True
$(((A \rightarrow B) \rightarrow A) \rightarrow A)$
$(((F \rightarrow T) \rightarrow F) \rightarrow F)$
$(((T) \rightarrow T) \rightarrow T)$
$((T) \rightarrow T)$
T

V_4
A = False
B = False
$(((A \rightarrow B) \rightarrow A) \rightarrow A)$
$(((F \rightarrow F) \rightarrow F) \rightarrow F)$
$(((T) \rightarrow T) \rightarrow T)$
$((T) \rightarrow T)$
T

2.
For (a), if any of a_n are true, the statement will be false. In other words, *all* of a_n must be false to return$_1$

true. This is the same as saying a_1 *and* a_2 *and* a_n must all be false to return true.

For (b), *any* of a_n need to be false for the statement to be true. In other words, $\sim a_1$ *or* $\sim a_2$ *or* $\sim a_n$ need to be false.

3.
It is proven in Example 2.9 that any statement can be converted into DNF. $a_1, a_2, \ldots a_n$ are conjunctive statements. A is the disjunction of $a_1, a_2, \ldots a_n$ and is thus in DNF form: $a_1 \vee a_2 \vee \ldots a_n$. If we take ~A, then we will get $\sim(a_1 \vee a_2 \vee \ldots a_n)$ which is equivalent, by De Morgan's law, to $\sim a_1 \wedge \sim a_2 \wedge \ldots \sim a_n$. We can then apply the distributive law to turn it back into DNF: $\sim b_1 \vee \sim b_2 \vee \ldots \sim b_n$. If we then negate that again, we get $\sim(\sim b_1 \vee \sim b_2 \vee \ldots \sim b_n)$, and applying De Morgan's law a second time, we finally get $b_1 \wedge b_2 \wedge \ldots b_n$. Thus, any statement can be turned into DNF, which can be turned into CNF.