August 14, 2020

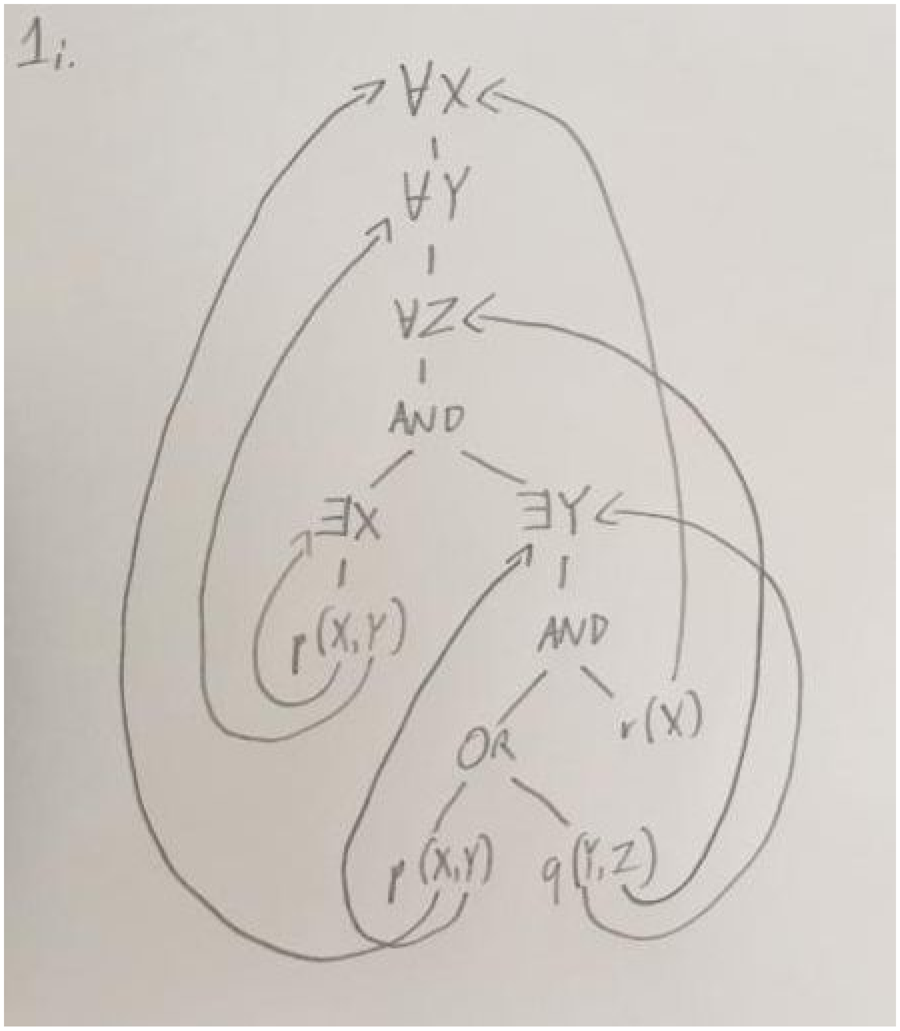
CS360 Quiz 4

Group members: Shivanshi Nagar, Stephen Hansen, Dennis George

Start Time 1:00 pm
End Time 2:30 pm

Question 1
i. Draw its expression tree and for each bound variable mark the corresponding bounding quantifier.

See expression tree pictured below. There is an arrow from each bound variable that points to the corresponding bounding quantifier (per week 7 figure 1).



ii. Bring it into prenex form. Justify each step of your transformation.
$(\forall X)((\forall Y)((\forall Z)((\exists X)p(X,Y)\ AND\ (\exists Y)((p(X,Y)\ OR\ q(Y,Z))\ AND\ r(X)))))$
Rename the inner variable quantifier X to W for $(\exists X)p(X,Y)$ since this is not the same X as the outermost $\forall X$ quantifier.
$(\forall X)((\forall Y)((\forall Z)((\exists W)p(W,Y)\ AND\ (\exists Y)((p(X,Y)\ OR\ q(Y,Z))\ AND\ r(X)))))$
Distribute the $(\exists W)$ over the entire AND expression as W is not free in the second clause (W is not present in the second clause, so moving the exists statement out does not affect the behavior of this clause.
$(\forall X)((\forall Y)((\forall Z)((\exists W)(p(W,Y)\ AND\ (\exists Y)((p(X,Y)\ OR\ q(Y,Z))\ AND\ r(X))))))$

Rename the inner variable quantifier Y to V for $(\exists Y)((p(X,Y)\ OR\ q(Y,Z))\ AND\ r(X))$ since this is not the same Y as the outermost $\forall Y$ quantifier.

$(\forall X)((\forall Y)((\forall Z)((\exists W)(p(W,Y)\ AND\ (\exists V)((p(X,V)\ OR\ q(V,Z))\ AND\ r(X))))))$

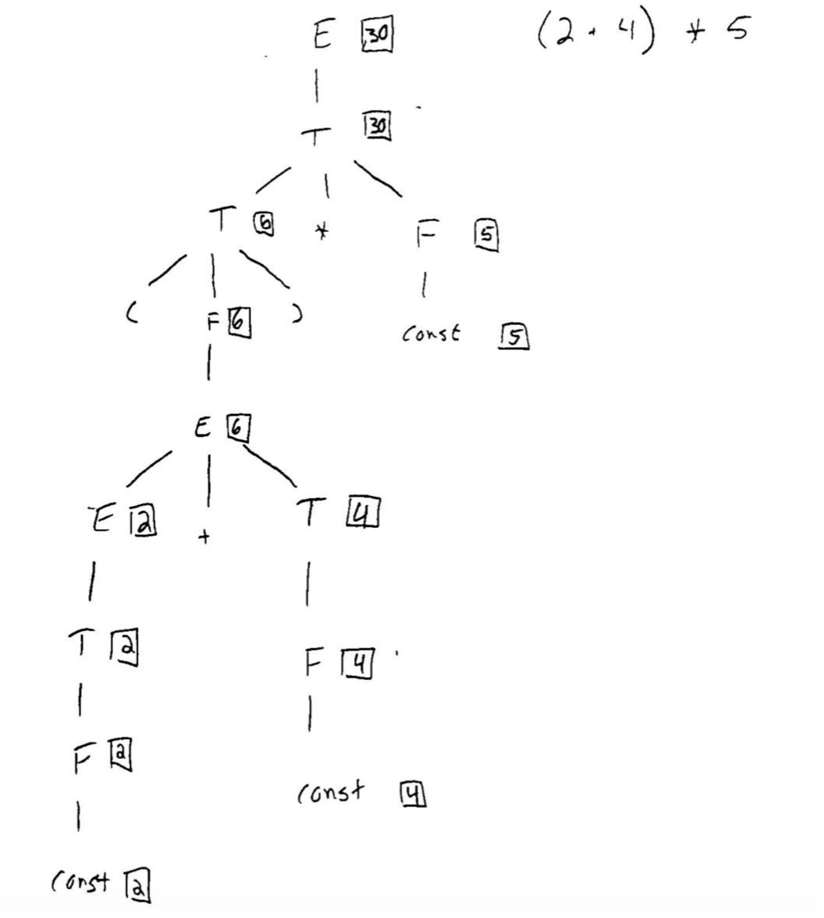
Distribute the $(\exists V)$ over the entire AND expression as V is not free in the first clause (V is not present in the first clause, so moving the exists statement out does not affect the behavior of this clause.

$(\forall X)((\forall Y)((\forall Z)((\exists W)((\exists V)(p(W,Y)\ AND\ ((p(X,V)\ OR\ q(V,Z))\ AND\ r(X)))))))$

We now arrive at a statement which is in prenex form as all quantifiers are grouped together in series and all quantifiers apply to the innermost Boolean expression (the expression itself contains no additional quantifiers). This concludes the conversion to prenex form and this statement is equivalent to the original logical expression.

Question 2
Explain the principal ingredients of the semantics of the attribute grammar of PLP Figure 4.1. What are semantic rules? Provide illustrating examples to your explanations.

The attribute grammar associates a *val* attribute with each *E,T,F,* and *const* in the grammar. This is so that for any symbol S, S.val will be the meaning, as an arithmetic value, of the token string derived from S. In figure 4.1, the *val* attributes are derived from the semantic rules on the right hand side. The semantic rules in figure 4.1 are simply familiar arithmetic operations, however they can be arbitrarily complex functions designed by the language designer. Figure 4.1 uses the semantic functions *sum*, *difference, product, quotient,* and *additive_inverse*. These functions are set to the *val* attribute of each node in the parse tree. In general semantic rules are either static or dynamic. Static rules are forced at compile-time, and dynamic rules are dynamic rules at run-time, as they cannot be caught during compilation. The semantic analyzer will force all of the semantic rules.
Using the example attribute grammar in figure 4.1, below is an illustration of a decorated parse tree for the expression (2 + 4) * 5. The boxes represent the *val* of the node, which is a result of the semantic rule.



The attributes flow strictly upward in this case (hence the grammar is S-attributed and attributes are synthesized, i.e. values are calculated only in productions when a symbol appears on the left hand side). Each box holds the output of a single semantic rule. For example, at the second level of the tree, the value 30 is the result of the application of the rule:
T1.val := product(T2.val, F.val)

Question 3

Note - *length* function computes number of digits in the number. May be implemented as a simple calculation of floor(log10(num)) + 1, for instance. Note that -1 has length 0 (we treat a negative number as a base case, i.e. an empty number, to handle the epsilon case). The grammar doesn't technically parse for sign (positive/negative), but adding a sign parse would be trivial as just checking the first character and multiplying the result of C by -1 if the number is negative.

- *positive?* Will return 0 if the *val* is negative. Otherwise return *val.* This is how we are handling epsilon, as it cannot be represented as a digit (treating all values as strictly numbers). Thus our grammar accumulates the value of a constant into C.val and is S-attributed (always moves the value up the parse tree). The epsilon case is handled with a special value "-1" which returns a length of 0 and positive value of 0 such that the recursive computation of digits.val may add each digit correctly to the final floating point number.

C -> digits_1 . digits_2
 ▷ C.val := digits_1.val + digits_2.val / 10^(*length*(digits_2.val))
digits -> digit more_digits
 ▷ digits.val := digit.val * 10^(*length*(more_digits.val)) + *positive?*(more_digits.val)
more_digits-> digits
 ▷ more_digits.val := digits.val
more_digits-> epsilon
 ▷ more_digits.val := -1
digit -> 0
 ▷ digit.val := 0
digit -> 1
 ▷ digit.val := 1
digit -> 2
 ▷ digit.val := 2
digit -> 3
 ▷ digit.val := 3
digit -> 4
 ▷ digit.val := 4
digit -> 5
 ▷ digit.val := 5
digit -> 6
 ▷ digit.val := 6
digit -> 7
 ▷ digit.val := 7
digit -> 8
 ▷ digit.val := 8
digit -> 9
 ▷ digit.val := 9

Question 4

Given the Horn clause resolution, of the following form with conditions:

$a \leftarrow a_1, a_2, \cdots, a_n$

$b \leftarrow b_1, b_{2,} \cdots, b_{i-1}, a, b_{i+1}, \cdots, b_m$

And conclusion:

$b \leftarrow b_1, b_{2,} \cdots, b_{i-1}, a_1, a_2, \cdots, a_n, b_{i+1}, \cdots, b_m$

We know that $a \leftarrow a_1, a_2, \cdots, a_n$ is the same statement as $a_1 a_2 \cdots a_n \rightarrow a$ and $b \leftarrow b_1, b_{2,} \cdots, b_{i-1}, a, b_{i+1}, \cdots, b_m$ is the same statement as $b_1 b_2 \cdots b_{i-1} a b_{i+1} \cdots b_m \rightarrow b$ (the commas signified a logical AND, which is represented here as multiplication and we switched the left and right hand sides of the implies statements). These implies statements may be changed to equivalent logical OR statements $\overline{a_1 a_2 \cdots a_n} + a$ and $\overline{b_1 b_2 \cdots b_{i-1} a b_{i+1} \cdots b_m} + b$. We may distribute the negation for the second statement as a negation over OR, such that the second statement becomes $\overline{b_1} + \overline{b_2} + \cdots + \overline{b_{i-1}} + \bar{a} + \overline{b_{i+1}} + \cdots + \overline{b_m} + b$. Since this statement is now a series of ORs, and OR is commutative, we may switch the places of a and b to get $\overline{b_1} + \overline{b_2} + \cdots + \overline{b_{i-1}} + b + \overline{b_{i+1}} + \cdots + \overline{b_m} + \bar{a}$. Since $b = \bar{\bar{b}}$ (double negation of itself), then the second equation may be rewritten as $\overline{b_1} + \overline{b_2} + \cdots + \overline{b_{i-1}} + \bar{\bar{b}} + \overline{b_{i+1}} + \cdots + \overline{b_m} + \bar{a}$. We may undo the distribution of the negation on OR for all elements except a to get $\overline{b_1 b_2 \cdots b_{i-1} \bar{b} b_{i+1} \cdots b_m} + \bar{a}$. We may again use the commutative property of OR with both statements to arrive at two conditions $a + \overline{a_1 a_2 \cdots a_n}$ and $\bar{a} + \overline{b_1 b_2 \cdots b_{i-1} \bar{b} b_{i+1} \cdots b_m}$.

For the conclusion, $b \leftarrow b_1, b_{2,} \cdots, b_{i-1}, a_1, a_2, \cdots, a_n, b_{i+1}, \cdots, b_m$ is the same statement as $b_1 b_2 \cdots b_{i-1} a_1 a_2 \cdots a_n b_{i+1} \cdots b_m \rightarrow b$, hence by rewriting as a logical OR we get $\overline{b_1 b_2 \cdots b_{i-1} a_1 a_2 \cdots a_n b_{i+1} \cdots b_m} + b$. Distributing the negation over AND yields $\overline{b_1} + \overline{b_2} + \cdots + \overline{b_{i-1}} + \overline{a_1} + \overline{a_2} + \cdots + \overline{a_n} + \overline{b_{i+1}} + \cdots + \overline{b_m} + b$. We may swap the positions of the sum of a terms and the b term (commutative OR) to get $\overline{b_1} + \overline{b_2} + \cdots + \overline{b_{i-1}} + b + \overline{b_{i+1}} + \cdots + \overline{b_m} + \overline{a_1} + \overline{a_2} + \cdots + \overline{a_n}$. The negation of a statements may be regrouped as a distributed negation over an AND of the statements, giving $\overline{b_1} + \overline{b_2} + \cdots + \overline{b_{i-1}} + b + \overline{b_{i+1}} + \cdots + \overline{b_m} + \overline{a_1 a_2 \cdots a_n}$. Again, rewrite the b term to be a double negation and distribute the negation over all b terms (like we did for the second condition) to get $\overline{b_1 b_2 \cdots b_{i-1} \bar{b} b_{i+1} \cdots b_m} + \overline{a_1 a_2 \cdots a_n}$. Flip the two sides (commutative OR) to get $\overline{a_1 a_2 \cdots a_n} + \overline{b_1 b_2 \cdots b_{i-1} \bar{b} b_{i+1} \cdots b_m}$.

We see with conditions $a + \overline{a_1 a_2 \cdots a_n}$ and $\bar{a} + \overline{b_1 b_2 \cdots b_{i-1} \bar{b} b_{i+1} \cdots b_m}$, and conclusion $\overline{a_1 a_2 \cdots a_n} + \overline{b_1 b_2 \cdots b_{i-1} \bar{b} b_{i+1} \cdots b_m}$ that we may choose $p = a$, $q = \overline{a_1 a_2 \cdots a_n}$, and $r = \overline{b_1 b_2 \cdots b_{i-1} \bar{b} b_{i+1} \cdots b_m}$ to arrive precisely at the FCS resolution having conditions $p + q$ and $\bar{p} + r$ and conclusion $q + r$, thus proving that the Horn clause resolution implies the FCS resolution.