Introduction to Computers and Programming

Prof. I. K. Lundqvist

Recitation 5
May 13 2004

Logic in proofs

Rule of Inference	Tautology	Name
p $\therefore p \lor q$	$p \rightarrow (p \lor q)$	Addition
$p \land q$ $\therefore p$	$(p \land q) \rightarrow p$	Simplification
p, q $\therefore p \land q$	$(p \land q) \rightarrow p \land q$	Conjunction
$p, p \rightarrow q$ $\therefore q$	$(p \land (p \rightarrow q)) \rightarrow q$	Modus Ponens
$\neg q, p \rightarrow q$ $\therefore \neg p$	$(\neg q \land (p \rightarrow q)) \rightarrow \neg p$	Modus Tollens
$p \rightarrow q, q \rightarrow r$ $\therefore p \rightarrow r$	$((p \rightarrow q) \land (q \rightarrow r)) \rightarrow (p \rightarrow r)$	Hypothetical Syllogism
$p \lor q, \neg p$ $\therefore q$	$((p \lor q) \land \neg p) \rightarrow q$	Disjunctive Syllogism
$p \lor q, \neg p \lor r$ $\therefore q \lor r$	$(p \lor q) \land (\neg p \lor r) \rightarrow q \lor r$	Resolution

Rules of Inference

- Addition

$$\frac{p}{\therefore p \vee q}$$

  - RedSox will win — p
  - RedSox or the Mets will win — $p \vee q$

- Simplification

$$\frac{p \wedge q}{\therefore p}$$

  - RedSox will win and The Yankees will not — $p \wedge q$
  - RedSox will win — p

Rules of Inference

- Conjunction

$$\begin{array}{r} p \\ q \\ \hline \therefore p \wedge q \end{array}$$

  - RedSox will win — p
  - The Yankees will loose — q
  - The RedSox will win and the Yankees will loose — $p \wedge q$

Rules of Inference

- Modus Ponens

$$\begin{array}{r} p \rightarrow q \\ p \\ \hline \therefore q \end{array}$$

  - If it is raining or snowing, the ground is wet $(R \vee S) \rightarrow W$
  - It is raining or snowing $(R \vee S)$
  - The ground is wet W

Rules of Inference

- Modus Tollens

$$\begin{array}{r} p \rightarrow q \\ \neg q \\ \hline \therefore \neg p \end{array}$$

  - If it is raining or snowing, the ground is wet $(R \vee S) \rightarrow W$
  - The ground is not wet $\neg W$
  - It is not raining nor snowing $\neg (R \vee S)$

Rules of Inference

- Hypothetical syllogism

$$\begin{array}{r} p \rightarrow q \\ q \rightarrow r \\ \hline \therefore p \rightarrow r \end{array}$$

  - If it is raining, the ground is wet $p \rightarrow q$
  - If the ground is wet, use an umbrella $q \rightarrow r$
  - If it is raining, use an umbrella

Rules of Inference

- Disjunctive syllogism

$$\begin{array}{r} p \vee q \\ \neg p \\ \hline \therefore q \end{array}$$

  - It is either snowing or raining $p \vee q$
  - It is not snowing $\neg p$
  - It is raining q

Rules of Inference

- Resolution

$$\begin{array}{r} p \vee q \\ \neg p \vee r \\ \hline \therefore q \vee r \end{array}$$

  – It is snowing or raining P v Q
  – It is not snowing or hale ¬P v R
  – It is raining or hale Q v R)

Rules of Inference

- Constructive dilemma

$$\begin{array}{r} p \vee q \\ p \rightarrow r \\ q \rightarrow s \\ \hline \therefore r \vee s \end{array}$$

  – Either RedSox or Yankees will win P v Q
  – If RedSox wins, then Boston goes wild P → R
  – If Yankees wins, then NYC goes wild Q → S
  – Boston or NYC goes wild R v S

Example

- Prove that
  - [(P ∨ Q) → R] ∧ [R → (S → T)] ∧ [P ∧ S] → T
  - [(A ∧ B) ∨ ¬C] ∧ [(A ∧ B) → D] ∧ [E ∨ ¬D] ∧ ¬E → ¬C
  - [(¬I ∧ J) → K] ∧ [¬L → J] ∧ [¬L ∧ ¬I] → K ∨ M

Simple Exception Handling

```
function Tan (
        X : Float )
    return Float is
  begin
     return Sin(X) / Cos(X);
  exception
     when Numeric_Error =>
        if (Sin(X)>=0.0 and Cos(X)>= 0.0) or
           (Sin(X)< 0.0 and Cos(X)<= 0.0) then
           return Float'Last;
        else
           return -Float'Last;
        end if;
  end Tan;
```

Exception Handling

Exception

```
procedure Safe_Get_Float(
        Out_Float :    out Float;
        Min, Max  : in     Float   ) is

     Local_Float : Float;
     Good_One    : Boolean := False;

begin -- Safe_Get_Float
   while not Good_One loop
      begin
         Put("Enter a float in range ");
         Put( Min, Exp => 0 );
        Put( " to ");
        Put( Max, Exp => 0 );
         Put( " ");
         Get( Local_Float );
         -- this point can only be
         -- reached if the get
         -- did not raise the exception

         -- now tested against limits
         -- specified
         Good_One:=((Local_Float>=Min) and
                    (Local_Float<=Max));

         if not Good_One then
           raise Data_Error;
           -- Loal_Float < Min OR
           -- Local_Float > Max
        end if;
exception
   when Data_Error =>
      Put_Line("DATA ERROR. Invalid
               input, pls try again ");
      new_line;
      Skip_Line;
   end; -- protected block of code

   end loop;
      -- this point can only be reached
      -- when valid value input
      Skip_Line;

      Out_Float := Local_Float;
      -- export input value
   end Safe_Get_Float;
```

Conventional Execution

Exception Handling

Exception

```
procedure Safe_Get_Float(
        Out_Float :    out Float;
        Min, Max  : in     Float   ) is

     Local_Float : Float;
     Good_One    : Boolean := False;

begin -- Safe_Get_Float
   while not Good_One loop
      begin
         Put("Enter a float in range ");
         Put( Min, Exp => 0 );
        Put( " to ");
        Put( Max, Exp => 0 );
         Put( " ");
         Get( Local_Float );
         -- this point can only be
         -- reached if the get
         -- did not raise the exception

         -- now tested against limits
         -- specified
         Good_One:=((Local_Float>=Min) and
                    (Local_Float<=Max));

         if not Good_One then
            raise My_Error;   <--
            -- Loal_Float < Min OR
            -- Local_Float > Max
         end if;
 exception
   when Data_Error =>
    Put_Line("DATA ERROR. Invalid
             input, pls try again ");
    new_line;
    Skip_Line;
  when My_Error =>
    Put_Line("MY ERROR. Invalid input,
             pls try again");
    new_line;
    skip_Line;
    raise My_Error;   <--
 end; -- protected block of code
 end loop;
      -- this point can only be
      -- reached when valid value input
      Skip_Line;

      Out_Float := Local_Float;
      -- export input value
end Safe_Get_Float;
```

Conventional Execution

Infix Evaluation

- Check if the parentheses are balanced
- Parse the input string from left to right
  - If Input(I) is an operand, push it on operand stack
  - If Input(I) is an operator, push it on operator stack
  - If Input(I) = ‘)’
    - Pop two elements from the operand stack
    - Pop the operator from the operator stack
    - Perform computation and Push result back onto operator stack
- The value of the expression is now on top of operand stack

Binary Tree

A binary tree is a tree that is

1. Empty
2. Has two children left, right which are themselves binary trees

Prove that the height of a non-empty binary tree is at least $\lfloor \lg(n) \rfloor$, where n is the number of nodes in the tree.

Proof

- Given:
  - height = 1 + max (height of subtrees)
  - $\lfloor lg(n) \rfloor = \lfloor lg(n)-1 \rfloor$ if n>2 and n odd
- To prove:

  height(Tree) >= $\lfloor lg(num\_nodes) \rfloor$

Base Case

- For a tree with just the root node (n=1), the theorem holds lg(1) = 0

Inductive Step

- Assume n >= 2, theorem holds for $1 \leq j < n$
- Prove that theorem holds for j = n

Given that n>=2, the tree T can be split into two subtrees T_L and T_R

Assume that both T_L and T_R have equal number of nodes.

Inductive Step

$\lceil (n-1)/2 \rceil \leq T_L \leq \lfloor n-1 \rfloor$

Given that theorem holds for subtrees,
Height $(T_L) \geq \lg \lceil (n-1)/2 \rceil$

$\rightarrow$ Height(T)
$$\begin{aligned} &\geq \lfloor \lg \lceil (n-1)/2 \rceil \rfloor + 1 \\ &\geq \lfloor 1 + \lg \lceil (n-1)/2 \rceil \rfloor \\ &\geq \lfloor \lg (2\lceil (n-1)/2 \rceil) \rfloor \\ &\geq \lfloor \lg (n) \rfloor \end{aligned}$$