Exercise 1: Critical Pairs (Past exam question)

Find a non-trivial critical pair of the following pair of rewrite rules, where $x, y, u, v, w$ are variables:

$$s(x) + y \Rightarrow x + s(y) \text{ and } (u + v) + w \Rightarrow u + (v + w)$$

Explain your reasoning.

Exercise 2: Confluence (Past exam question)

Consider the following set of rewrite rules, where $X, Y, Z$ are variables:

$$
\begin{align*}
(X \cdot Y) \cdot Z & \Rightarrow X \cdot (Y \cdot Z) \\
X \cdot 0 & \Rightarrow X \\
X \cdot i(X) & \Rightarrow 0
\end{align*}
$$

Give two ways in which this system of rewrite rules is not locally confluent. Explain your answer in terms of critical pairs.

Exercise 3: Pen-and-Paper Hoare Logic Proof Exercise

Construct the natural deduction proof of the following Hoare Logic triple (taken from the factorial example in the lecture), on paper:

$$
\{ Y = 1 \land Z = 0 \} \text{WHILE } Z \neq X \text{ DO } Z := Z + 1 ; Y := Y \times Z \text{ OD } \{ Y = X! \}
$$

You may use any of the FOL natural deduction rules and the Hoare Logic rules. Additionally, you may use the following 4 lemmas:

$$
\begin{align*}
\neg \neg X & \Rightarrow X \text{ notnotD} \\
b = a & \Rightarrow a = b \text{ sym} \\
0! = 1 & \Rightarrow ! \text{ fact_0} \\
X! \times (x + 1) & = (x + 1)! \text{ fact_plus_1} \\
s = t & \Rightarrow P_s \text{ subst}
\end{align*}
$$
Exercise 4: Introduction to Hoare Logic in Isabelle

In the following exercise, you will formally verify the correctness of some simple programs using Isabelle's Hoare Logic library. This library allows you to formalise the specifications of programs of a simple programming language in the form of Hoare triples.

The supported programming language includes the following constructs:

- Local variable declaration: `VARS x y z`
- Sequence: `p ; q`
- Skip (do nothing): `SKIP`
- Variable assignment: `x := 0`
- Conditional: `IF cond THEN p ELSE q FI`
- Loop: `WHILE cond INV { invariant } DO p OD`

A program $X$ with precondition $P$ and postcondition $Q$ can be specified as the Hoare triple:

Each Hoare triple in Isabelle must begin with a local variable declaration `VARS` including at least one local variable, i.e. the triple shown above can be specified in Isabelle as follows:

"VARS a P X Q"

Note that a loop invariant must be explicitly specified for each while loop using the `INV` operator.

You can use any pre-defined Isabelle type or function in the program specification.

The automated tactic `vcg`, can be used to extract verification conditions from the Hoare triples and convert them to Isabelle subgoals. The tactic `vcg_simp` combines the capabilities of `vcg` with simplification.

Verification Problems

Using the Hoare Logic library, verify the correctness of the following programs. Some of the examples require that you introduce the appropriate invariant Inv.

Make sure your theory has the following imports statement:

`imports Main Binomial "~~/src/HOL/Hoare/Hoare Logic"`

- The minimum of two integers $x$ and $y$:

  `lemma Min: "VARS (z :: int)
  \{True\}
  IF x ≤ y THEN z := x ELSE z := y FI
  \{ z = min x y \}"

2
• Iteratively copy an integer variable $x$ to $y$:

```plaintext
lemma Copy: "VARS (a :: int) y
{0 ≤ x}
a := x; y := 0;
WHILE a ≠ 0
INV { Inv }
DO y := y + 1 ; a := a - 1 OD
{x = y}"
-- "Replace Inv with your invariant."
```

• Iterative multiplication through addition:

```plaintext
lemma Multi: "VARS (a :: int) z
{0 ≤ y}
a := 0; z := 0;
WHILE a ≠ y
INV { Inv }
DO
    z := z + x ;
    a := a + 1
OD
{z = x * y}"
-- "Replace Inv with your invariant."
```

• A factorial algorithm:

```plaintext
lemma DownFact: "VARS (z :: nat) (y::nat)
{True}
z := x; y := 1;
WHILE z > 0
INV { Inv }
DO
    y := y * z ;
    z := z - 1
OD
y = fact x"
-- "Replace Inv with your invariant."
```

• Integer division of $x$ by $y$:

```plaintext
lemma Div: "VARS (r :: int) d
{y ≠ 0}
r := x; d := 0;
WHILE y ≤ r
INV { Inv }
DO
    r := r - y;
    d := d + 1
OD
{ Postcondition }"
```
Exercise 5: Induction

Consider the following (Isabelle) datatype definition:

\[
\text{datatype } 'a \text{ TREE} = \text{LEAF } 'a | \text{NODE } 'a \ ' 'a \text{ TREE} \ ' 'a \text{ TREE}
\]

1. Give an induction rule appropriate for proofs by (structural) induction involving the \text{TREE} datatype.

2. Define a function \textit{MIRROR} that recursively flips the nodes in the left and right subtrees of a tree as defined above.

3. Formalize your definition of \textit{MIRROR} in Isabelle and give a structured (Isar) proof that \texttt{MIRROR(MIRROR \ t) = \ t}.