The following exercises are typical exam exercises. You are supposed to solve them on a sheet of paper, without using Isabelle/HOL.

**Exercise 13.1  Verification Condition Generation**

Regard the following While-program S:

\[
\begin{align*}
a & := x; \\
\text{WHILE } 1 < a & \text{ DO} \\
a & := a - 2
\end{align*}
\]

Your task is to show that:

\[
\models \{ x \geq 0 \} \ S \{ a = 0 \implies \text{even } x \}
\]

Find an invariant for the loop. Let \( S_{annot} \) be the annotated program, and \( Q := \{ a = 0 \implies \text{even } x \} \) be the postcondition. Which proof obligations result when using the verification condition generator? What does \( vc \ S_{annot} Q \) and \( pre \ S_{annot} Q \ s \) look like?

**Exercise 13.2  Parity analysis**

Now regard the following While-program:

\[
\begin{align*}
r & := 11; \\
a & := 11 + 11; \\
\text{WHILE } 1 < a & \text{ DO} \\
r & := r + 1 \\
a & := a - 2; \\
r & := a + 1
\end{align*}
\]

Add annotations for parity analysis to this program, and iterate the \( step' \)-function until a fixed point is reached. Document the results of each iteration in a table. Hint: Unlike sheet 12, you need to push the top-value of the lattice into the step function on each iteration!
Exercise 13.3 Abstract Interpretation For Conditionals

(To be done with Isabelle)
Regard the locale Val_abs. Define, analogous to plus', a function less' :: 'av ⇒ 'av ⇒ bool option that approximates less expressions: Some b means, the result is definitely b, and None means unknown. Insert also an appropriate assumption gamma_less' to the locale.

Then define a function bval' :: bexp ⇒ 'av st ⇒ bool option in the locale Abs_Int_Fun (analogous to aval'), and show a lemma bval_sound (analogous to aval'_sound).
Note: You are not required to modify the step' function.

Homework 13 Abstract Interpretation: Sign Analysis

Submission until Wednesday, 8 February 2012, 12:00 (noon).

In this homework assignment, you must use the abstract interpretation framework (theory file Abs_Int0.thy) to create a sign analysis: For each program variable, this will calculate which signs (positive, negative, or zero) it could possibly have. (Refer to Abs_Int0_parity.thy to see a similar analysis for evenness/oddness. You may want to use that theory as a template for this assignment.)

First, define a type sign to formalize the 8-element complete lattice shown here. The elements NEG, ZERO, and POS indicate variables that are definitely known to be negative, zero, or positive, respectively. The other elements represent combinations of these.

```
ANY
  ┌───────┐
  │      │
  │  ☐    │
  │      │
  └───────┘
  ☐     ☐
 /     /  \
/     /    /
NON-POS NON-ZERO NON-NEG
  │    │    │
  │ ☐  ☐  ☐
  │    │    │
  └─┐   └─┐   └─┐
NEG ZERO POS
  │    │    │
  │ ☐  ☐  ☐
  │    │    │
  └─┐   └─┐   └─┐
  ☐     ☐   ☐
  \     \  /
   \     \
    NONE
```

One approach is to formalize sign as an 8-constructor datatype. But note that other representations are also possible!

Next, instantiate the preord and SL_top type classes: Define the ordering (op ⊑), join operator (op ⊔), and top element (⊤), and prove that they satisfy the class axioms.

```
instantiation sign :: preord
begin
  fun le_sign :: "sign ⇒ sign ⇒ bool" where
    instance
end
```
In order to instantiate the `Val_abs` and `Abs_Int` locales, you must first define three functions that describe the meaning of the `sign` type. The function $\gamma_{\text{sign}}$ yields the set of possible integer values that correspond to each `sign`. For example, when applied to the value representing `NON_NEG`, it should return a set equal to $\{i \mid 0 \leq i\}$.

```
fun $\gamma_{\text{sign}} :: \text{sign} \Rightarrow \text{val set}$ where
```

The function $\text{num}_{\text{sign}}$ returns the most specific `sign` value that includes the given integer: `NEG`, `ZERO`, or `POS`, as appropriate.

```
fun $\text{num}_{\text{sign}} :: \text{val} \Rightarrow \text{sign}$ where
```

The `plus_{sign}` function performs addition on `sign` values. It should always return the most specific element possible. For example, `NON_NEG + POS = POS`, and `NEG + POS = ANY`.

```
fun $\text{plus}_{\text{sign}} :: \text{sign} \Rightarrow \text{sign} \Rightarrow \text{sign}$ where
```

Now instantiate the `Val_abs` and `Abs_Int` locales. The `Val_abs` locale requires you to supply some proofs, while `Abs_Int` does not.

```
interpretation \text{Val_abs} where $\gamma = \gamma_{\text{sign}}$ and $\text{num}' = \text{num}_{\text{sign}}$ and $\text{plus}' = \text{plus}_{\text{sign}}$
```

```
interpretation \text{Abs_Int} where $\gamma = \gamma_{\text{sign}}$ and $\text{num}' = \text{num}_{\text{sign}}$ and $\text{plus}' = \text{plus}_{\text{sign}}$
defines $\text{aval}_{\text{sign}}$ is $\text{aval}'$ and $\text{step}_{\text{sign}}$ is $\text{step}'$ and $\text{AI}_{\text{sign}}$ is $\text{AI}$
```

Define and test the following example program as shown here. What does the analysis tell you about the values of $x$ and $y$?

```
definition \text{test1}_{\text{sign}} =
  "x" ::= \text{N} 0;
  "y" ::= \text{Plus} (V "x") (\text{N} 1);
  \text{WHILE} \text{Less} (V "x") (\text{N} 10) \text{DO}
  "x" ::= \text{Plus} (V "x") (\text{N} 2);
  "y" ::= \text{Plus} (V "x") (V "y")"
```

```
value \text{showacom_opt} (\text{AI}_{\text{sign}} \text{test1}_{\text{sign}})
```

Finally, you must define a measure function for type `sign`, which can be used to prove that the analysis always terminates. Define a function $m_{\text{sign}}$ and show that it satisfies the following two properties.
fun m_sign :: "sign ⇒ nat" where

lemma m_sign_gt: "[x ⊆ y; ¬ y ⊆ x] ⇒ m_sign x > m_sign y"

lemma m_sign_eq: "[x ⊆ y; y ⊆ x] ⇒ m_sign x = m_sign y"