Instructions

This exam contains 10 pages, including this one. Make sure you have all the pages. Write your name on the top of this page before starting the exam.

Write your answers on the exam sheets. If you finish at least 15 minutes early, bring your exam to the front when you are finished; otherwise, wait until the end of the exam to turn it in. Please be as quiet as possible.

If you have a question, raise your hand. If you feel an exam question assumes something that is not written, write it down on your exam sheet. Barring some unforeseen error on the exam, however, you shouldn’t need to do this at all, so be careful when making assumptions.

<table>
<thead>
<tr>
<th>Question</th>
<th>Score</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>30</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>15</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>20</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>100</td>
</tr>
</tbody>
</table>
Question 1. Short Answer (20 points).

a. **(5 points)** List one advantage and one disadvantage of polymorphic variants (i.e., a term constructed with a backtick like 'Reg 42) compared to regular OCaml datatypes.

   **Answer:** One advantage is that different polymorphic variant constructors can be mixed together arbitrarily (as long as they have the same contents types), whereas constructors from different datatypes cannot be mixed. On the other hand, the OCaml type system necessarily cannot detect as many errors involving polymorphic variants, e.g., it cannot easily find non-exhaustive match cases.

b. **(5 points)** Briefly describe what an intermediate representation is.

   **Answer:** An intermediate representation is a program representation that is at a level of abstraction in between the program source code/abstract syntax tree and the compiler output (typically bytecode or machine code).
c. (5 points) Briefly explain the relationship between a partial order and a lattice.

Answer: A lattice is a partial order such that for every pair of elements $x$ and $y$, the meet $x \sqcap y$ and join $x \sqcup y$ exist.

d. (5 points) For $\alpha$ (an abstraction function) and $\gamma$ (a concretization function) to form a Galois insertion, both must be monotonic, and two other properties must hold. What are the other two properties?

Answer:

$S \subseteq \gamma(\alpha(S))$ for any concrete set $S$ and

$\alpha(\gamma(A)) = A$ for any abstract element $A$. 
Question 2. Boolean Expressions and Bitvectors (10 points). Recall the types for boolean expressions, assignments, and bitvectors from Project 1:

```
type bexpr =  
  EFalse  
  | ETrue  
  | EVar of string  
  | EAnd of bexpr * bexpr  
  | EOr of bexpr * bexpr  
  | ENot of bexpr  
  | EForall of string * bexpr  
  | EExists of string * bexpr  

| ENot of bexpr  
| EForall of string * bexpr  
| EExists of string * bexpr  

type asst = (string * bool) list  
type bvec = bexpr list (* low order bit at head of list *)
```

a. (5 points) Write a function `bexpr if (b1:bexpr) (b2:bexpr) (b3:bexpr):bexpr` that returns a boolean expression that evaluates to the value of `b2` if `b1` evaluates to `true` and to `b3` otherwise. For example, the following expressions both evaluate to `true`.

```
eval ["x", EFalse] (bexpr if ETrue ETrue EFalse)
eval ["x", EFalse; "y", ETrue] (bexpr if (EVar "x") EFalse (EVar "y"))
```

**Answer:**

```
let bexpr if b1 b2 b3 = EOr (EAnd(b1, b2), EAnd(ENot(b1), b2))
```

b. (5 points) Write a function `bvec if (b1:bexpr) (b2:bvec) (b3:bvec):bvec` that returns a bitvector where the bit at position `i` evaluates to the value of the `i`th bit of `b2` if `b1` evaluates to `true`, and to the `i`th bit of `b3` otherwise. You can call `bexpr if` as a subroutine, and you can assume `b2` and `b3` have the same length.

**Answer:**

```
List.map2 (bexpr if b1) b2 b3
```
Question 3. Data flow analysis (30 points).

a. (25 points) In the following table, show each iteration of very busy expressions for the control-flow graph on the right. For each iteration, list the statement taken from the worklist in that step, the value of in computed for that statement, and the new worklist at the end of the iteration. You may or may not need all the iterations; you may also add more iterations if needed. Do not add the exit node to the worklist. Assume that \( x < 42 \) is not a possible very busy expression.

Use \( \emptyset \) for the set of no expressions, and \( \top \) for the set of all expressions. What is \( \top \)?

\[
\top = \{a+b, x+3, x+t\}
\]

What are the initial \( \text{in}'s \) for each statement?

<table>
<thead>
<tr>
<th>Stmt</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial in</td>
<td>\top</td>
<td>\top</td>
<td>\top</td>
<td>\top</td>
<td>\top</td>
<td>\top</td>
<td>\top</td>
<td>\emptyset</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Iteration</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stmt taken from worklist</td>
<td>N/A</td>
<td>6</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>in of taken stmt</td>
<td>N/A</td>
<td>a+b</td>
<td>a+b</td>
<td>a+b, x+t</td>
<td>a+b</td>
<td>a+b</td>
<td>x+3, a+b</td>
<td>a+b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>New worklist</td>
<td>0,1,2,3,4,5,6</td>
<td>0,1,2,3,4,5</td>
<td>0,1,2,4,5</td>
<td>0,1,2,4</td>
<td>0,1,2,3</td>
<td>0,1,2</td>
<td>0,1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Iteration</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stmt taken from worklist</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>in of taken stmt</td>
<td>a+b</td>
<td>x+3, a+b</td>
<td>a+b</td>
<td>a+b</td>
<td>a+b</td>
<td>a+b</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>New worklist</td>
<td>0,1,2</td>
<td>0,1</td>
<td>0</td>
<td>\emptyset</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
b. **(5 points)** Write the gen and kill sets for *available expressions* for the statements shown below. Assume the set of expressions is \{a+b, a+1, x+y, x+1\}. Write \(\emptyset\) for an empty gen or kill set.

<table>
<thead>
<tr>
<th>stmt</th>
<th>gen</th>
<th>kill</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x := a + b)</td>
<td>(a+b)</td>
<td>(x+y, x+1)</td>
</tr>
<tr>
<td>(x := y)</td>
<td>(\emptyset)</td>
<td>(x+y, x+1)</td>
</tr>
<tr>
<td>(a := a + 1)</td>
<td>(\emptyset)</td>
<td>(a+b, a+1)</td>
</tr>
</tbody>
</table>
Question 4. Static Single Assignment Form (15 points). Draw a static single assignment form control-flow graph for the following program. Put all statements in the largest possible basic blocks. Be sure to include the entry and exit nodes. Don’t worry about whether Φ nodes go in their own basic blocks or get added to existing ones—do whichever is convenient.

```c
x = 13;
y = x + 3;
z = y - x;
while (z < y) {
    w = z - 1;
    if (w < y) {
        w = w + 2
    } else {
        w = w - 2
    }
y = w + 3
}
```

Answer:

```
x0 = 13
y0 = x0 + 3
z0 = y0 - x0
y1 = Φ(y0, y2)
z0 < y1
w0 = z0 - 1
w0 < y1
w3 = Φ(w1, w2)
y2 = w3 + 3
```
Question 5. Abstract Interpretation (20 points). In this problem, you will develop part of an abstract interpreter for RubeVM. The abstract domain will be the domain of intervals, where an interval $[x, y]$ represents all integers $n$ such that $x \leq n \leq y$. An interval is empty if $x > y$. We will represent an interval as a type $\text{inter} = \text{int} \times \text{int}$, where the left component is the lower bound and the right component is the upper bound.

Below is part of the RubeVM instruction set. We’ve modified the type of the register file to map registers to abstract values.

<table>
<thead>
<tr>
<th>Type of register file</th>
<th>Type of values</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reg (int)</td>
<td>Int (int)</td>
<td>$I\text{const}$ of reg * value ($<em>dst$, src</em>)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$I\text{add}$ of reg * reg * reg ($<em>dst$, src1, src2</em>)</td>
</tr>
</tbody>
</table>

a. (5 points) Implement an abstraction function $\alpha : \text{int} \rightarrow \text{inter}$ that returns the interval representing a single integer, and a concretization function $\gamma : \text{inter} \rightarrow \text{int list}$ that returns a list of the integers (in any order) corresponding to an interval.

Answer:

```plaintext
let alpha n = (n, n)
let rec gamma (x, y) = if x > y then [] else x :: (gamma (x+1, y))
```

b. (10 points) Write a function $\text{arun}\text{inst} : \text{regs} \rightarrow \text{instr} \rightarrow \text{unit}$ that abstractly executes the three instructions given in the figure above. **Hint:** Subtraction can be implemented as negation followed by addition. You will probably want to use Hashtbl.add and Hashtbl.find, with signatures:

```
val add : ('a, 'b) t -> 'a -> 'b -> unit
val find : ('a, 'b) t -> 'a -> 'b
```

open Hashtbl

Answer:

```plaintext
let empty (x, y) = y < x

let arun_inst rs = function
| I\text{const} ('Reg r, 'Int n) -> add(r, alpha n)
| I\text{add} ('Reg r1, 'Reg r2, 'Reg r3) ->
  let (x1, x2) = find rs r2 in
  let (y1, y2) = find rs r3 in
  if (empty (x1, x2)) || (empty (y1, y2)) then (1, 0)
  else add rs r1 (x1+y1, x2+y2)
| I\text{sub} ('Reg r1, 'Reg r2, 'Reg r3) ->
  let (x1, x2) = find rs r2 in
  let (y1, y2) = find rs r3 in
  if (empty (x1, x2)) || (empty (y1, y2)) then (1, 0)
  add rs r1 (x1−y2, x2−y1)
```
c. (5 points) As a step toward abstractly executing branches, implement a function \( \text{join} : \text{inter} \to \text{inter} \to \text{inter} \) that returns the smallest interval containing both of its arguments.

Answer:

\[
\begin{align*}
\text{let } \text{join} \ (x1, x2) \ (y1, y2) &= \\
&\quad \text{if } x2 < x1 \text{ then } (y1, y2) \\
&\quad \text{else if } y2 < y1 \text{ then } (x1, x2) \\
&\quad \text{else } (\min x1 y1, \max x2 y2)
\end{align*}
\]
Question 6. Subtyping (5 points). In class we discussed a subtyping system for the following language, which is the simply typed lambda calculus extended with tainted (\(n^t\) with type \(\text{int}^t\)) and untainted (\(n^u\) with type \(\text{int}^u\)) integers, where \(\text{int}^u \leq \text{int}^t\):

\[
\begin{align*}
e & ::= n^t \mid n^u \mid x \mid \lambda x : t . e \mid e \; e \\
t & ::= \text{int}^t \mid \text{int}^u \mid t \rightarrow t
\end{align*}
\]

Suppose we extend the language with pair types \(t \times t\) with the following subtyping rule:

\[
\begin{align*}
t_1 \leq t_2 & \quad t'_1 \leq t'_2 \\
(t_1 \times t'_1) \leq (t_2 \times t'_2)
\end{align*}
\]

Give a list of all types \(t\) such that \(t\) is a subtype of \(\text{int}^t \times (\text{int}^t \rightarrow \text{int}^u)\). Hint: It might be easiest to write down all potential subtypes and then cross out the ones that aren’t actually subtypes.

Answer:

\[
\begin{align*}
\text{int}^u \times (\text{int}^t \rightarrow \text{int}^u) \\
\text{int}^t \times (\text{int}^t \rightarrow \text{int}^u)
\end{align*}
\]