Final Exam

Functional Data Structures

15. 2. 2023

First name: ____________________

Last name: ____________________

Student-Id (Matrikelnummer): ____________________

Signature: ____________________

1. The exam is to be solved in Isabelle on your own Laptop.
2. You may use your own notes, as well as all lecture material to solve the exam.
3. Using the internet is not allowed for any other reason than submitting your solution to the submission system.
4. You have 120 minutes to solve the exam.
5. Please put your student ID and ID-card or driver's license on the table until we have checked it.
6. Please do not leave the room in the last 20 minutes of the exam — you may disturb other students who need this time.

Proof Guidelines: We expect valid Isabelle proofs. `sledgehammer` may be used. The use of `sorry` may lead to the deduction of points but is preferable to spending a lot of time on individual proof steps. Unfinished proofs should be well written and easy to understand!

1 Induction (Solution)

```
lemma count_app: "count(as @ bs) = count as + count bs"
proof (induction as)
  case (Cons a as)
  then show ?case
    by (cases a) auto
qed auto

lemma ok_count_Zero: "ok bs ⟹ count bs = 0"
  by (induction rule: ok.induct) (auto simp: count_app)
```

2 Bounds on Small-Step Execution (Solution)

```
fun bound :: "com ⇒ nat" where
"bound (IF _ THEN c1 ELSE c2) = 1 + max (bound c1) (bound c2)" |
"bound (c1 ;; c2) = 1 + bound c1 + bound c2" |
"bound (_ ::= _) = 1" |
"bound _ = 0"

lemma bound_decr: "while_free c ⟹ (c, s) → (c', s') ⟹ bound c' < bound c"
proof -
  assume "(c, s) → (c', s')" "while_free c" thus ?thesis
    by (induction rule: small_step_induct) auto
qed
```

3 Hoare Logic (Solution)

abbreviation (*input*) *change_rule* :: *"vname ⇒ bexp ⇒ assn ⇒ assn"* **where**
"change_rule x b P ≡ λs. ∀ n. bval b (s(x:=n)) ⟶ P(s(x:=n))"

lemma *sound*: "⊨ {*change_rule x b P*} *CHANGE x ST b* {*P*}"
  unfolding *hoare_valid_def* **by** *auto*

lemma *complete*: "⊨ {*P*} (*CHANGE x ST b*) {*Q*} ⟹ ⊢ {*P*} *CHANGE x ST b* {*Q*}"
  unfolding *hoare_valid_def*
  by(*rule strengthen_pre*[**where** *P="change_rule x b Q"*])
    (*auto intro*: *Change*)

abbreviation *"eq a1 a2 ≡ And (Not (Less a1 a2)) (Not (Less a2 a1))"*

definition *MINUS* :: *com* **where**
 "MINUS = CHANGE ''y'' ST eq (Plus (V ''x'') (V ''y'')) (N 0)"

lemma *MINUS_correct*:
  "⊢ {λs. $s{=}s_0$} *MINUS* {λs. *s ''y'' = − s ''x''* ∧ (∀ *v* ≠ *''y''*. *s v* = s_0 *v*)}"
  unfolding *MINUS_def*
  by (*rule strengthen_pre*[*OF _ Change*]) *auto*

definition *invar* :: *"vname ⇒ bexp ⇒ vname ⇒ state ⇒ assn"* **where**
  "invar x b y s_0 = (λs. *s x + s y* = s_0 *y* ∧ (∀ *v*. *v* ∉ {*x*,*y*} ⟶ *s v* = s_0 *v*))"

4 Abstract Proof (Solution)

```
lemma fixp:
  fixes f :: "nat ⇒ nat"
  assumes incr: "∀ n. n ≤ f n"
  assumes A: "A = {m. ∃ n. m = f n}"
  assumes max: "∃ m ∈ A. ∀ n ∈ A. n ≤ m"
  shows "∃ k∈A. f k = k"
proof –
  from max obtain m where "m ∈ A" "∀ n ∈ A. n ≤ m" by blast
  from A have "f m ∈ A" by blast
  with ‹∀ n∈A. n ≤ m› have "f m ≤ m" by blast
  with incr have "f m = m" by (simp add: le_antisym)
  with ‹m ∈ A› show ?thesis by auto
qed
```

5 Abstract Interpretation (Solution)

fun *less_eq_bits* :: *"bits ⇒ bits ⇒ bool"* (**infix** *"≤" 50*) **where**
"_ ≤ Any ⟷ True" |
"(B n) ≤ B m ⟷ n ≤ m" |
"(_::bits) ≤ _ ⟷ False"

fun *sup_bits* :: *"bits ⇒ bits ⇒ bits"* (**infix** *"⊔" 50*) **where**
"_ ⊔ Any = Any" |
"Any ⊔ _ = Any" |
"(B x) ⊔ (B y) = B (max x y)"

definition *"is_complete_lattice = True"*

As it is a total order over a nonempty set.

fun *plus'_bits* :: *"nat ⇒ bits ⇒ bits ⇒ bits"* **where**
"plus'_bits _ Any _ = Any" |
"plus'_bits _ _ Any = Any" |
"plus'_bits n (B x) (B y) = (if max x y = n then Any else B (Suc (max x y)))"

definition "A_0=[*None*,*Some*(*B 1*),______,______,______,______,______]"
definition "A_1=[*None*,______,*Some*(*B 1*),______,______,*Some*(*Any*),______]"
definition "A_2=[*None*,______,______,*Some*(*B 1*),______,______,*Some*(*Any*)]"
definition "A_3=[*None*,______,______,______,*Some*(*Any*),______,______]"
definition "A_4=[*None*,______,______,*Some*(*B 1*),______,______,*Some*(*Any*)]"

fun *inv_plus'_bits* :: *"bits ⇒ bits ⇒ bits ⇒ (bits * bits)"* **where**
"inv_plus'_bits _ (B 0) _ = (B 0, B 0)" |
"inv_plus'_bits _ _ (B 0) = (B 0, B 0)" |
"inv_plus'_bits Any x y = (x, y)" |
"inv_plus'_bits (B r) Any Any = (B r, B r)" |
"inv_plus'_bits (B r) (B a1) Any = (B (min r a1), B r)" |
"inv_plus'_bits (B r) Any (B a2) = (B r, B (min r a2))" |
"inv_plus'_bits (B r) (B a1) (B a2) = (B (min r a1), B (min r a2))"

fun *inv_less'_bits* :: *"bool ⇒ bits ⇒ bits ⇒ (bits * bits)"* **where**
"inv_less'_bits _ (B 0) _ = (B 0, B 0)" |
"inv_less'_bits _ _ (B 0) = (B 0, B 0)" |
"inv_less'_bits True Any (B n) = (B n, B n)" |
"inv_less'_bits _ Any x = (Any, x)" |
"inv_less'_bits False (B n) Any = (B n, B n)" |
"inv_less'_bits _ x Any = (x, Any)" |
"inv_less'_bits True (B a1) (B a2) = (B (min a1 a2), B a2)" |

"inv_less'_bits False (B a1) (B a2) = (B a1, B (min a1 a2))"