

Isabelle/HOL Exercises

Advanced

Interval Lists

```
types intervals = "(nat*nat)list"
```

```
consts inv2 :: "nat  $\Rightarrow$  intervals  $\Rightarrow$  bool"  
  set_of :: "intervals  $\Rightarrow$  nat set"  
  add1 :: "nat  $\Rightarrow$  intervals  $\Rightarrow$  intervals"  
  del1 :: "nat  $\Rightarrow$  intervals  $\Rightarrow$  intervals"
```

primrec

```
"inv2 j [] = True"  
"inv2 j (mn#ins) = (let (m,n) = mn in j <= m & m <= n & inv2 (n+2) ins)"
```

```
constdefs inv :: "intervals  $\Rightarrow$  bool"  
  "inv ins == inv2 0 ins"
```

primrec

```
"set_of [] = {}"  
"set_of (ij#ins) = {k. fst ij <= k & k <= snd ij} Un set_of ins"
```

primrec

```
"add1 i [] = [(i,i)]"  
"add1 i (jk#ins) = (let (j,k) = jk in  
  (if Suc i < j then (i,i)#(j,k)#ins else  
   if Suc i = j then (i,k)#ins else  
   if i <= k then jk#ins else  
   if i=Suc k then case ins of [] => [(j,i)]  
     | (m,n)#ins' =>  
       if m=Suc(Suc k) then (j,n)#ins' else (j,Suc k)#ins  
     else (j,k)#add1 i ins))"
```

primrec

```
"del1 i [] = []"  
"del1 i (jk#ins) = (let (j,k) = jk in  
  if i<j then jk#ins else  
  if i=j then if j=k then ins else (j+1,k)#ins else
```

```

    if i<k then (j,i - 1)#(i+1,k)#ins else
    if i=k then (j,k - 1)#ins
    else jk # del1 i ins)"

declare Let_def[simp] split_split[split]

lemma inv2_add1[rule_format]:
  "∀m. m ≤ i → inv2 m ins → inv2 m (add1 i ins)"
apply(induct ins)
  apply(simp)
apply(simp split: list.split)
done

theorem inv_add1: "inv ins ⇒ inv (add1 i ins)"
by(simp only:inv_def inv2_add1[of 0])

lemma set_of_add1[rule_format]:
  "∀m. inv2 m ins → set_of(add1 i ins) = insert i (set_of ins)"
apply(induct ins)
  apply(force)
apply(clarsimp)
apply(rule conjI)
  apply(simp split:list.split)
  apply(rule conjI)
  apply clarify
  apply(rule set_ext)
  apply (simp)
  apply arith
  apply clarify
  apply(rule conjI)
  apply clarify
  apply(rule set_ext)
  apply (simp)
  apply arith
  apply clarify
  apply(rule set_ext)
  apply (simp)
  apply arith
  apply clarify
  apply(rule conjI)
  apply clarify
  apply(rule conjI)
  apply clarify

```

```

    apply(rule set_ext)
    apply (simp)
    apply arith
  apply clarify
  apply(rule conjI)
    apply clarify
    apply(rule set_ext)
    apply (simp)
    apply arith
  apply clarify
  apply(rule set_ext)
  apply (simp)
  apply arith
  apply (fastsimp)
done

```

```

theorem "inv ins  $\implies$  set_of(add1 i ins) = insert i (set_of ins)"
by(simp only:inv_def set_of_add1[of 0])

```

```

lemma inv2_mono[rule_format]: "[ inv2 m ins; n<=m ]  $\implies$  inv2 n ins"
by(induct "ins", auto)

```

```

lemma inv2_del1[rule_format]: " $\forall m. inv2 m ins \longrightarrow inv2 m (del1 i ins)$ "
  apply(induct ins)
    apply(simp)
  apply(clarsimp)
  apply(rule conjI)
    apply clarsimp
    apply(rule conjI)
      apply(force intro:inv2_mono)
      apply(clarsimp)
      apply(rule conjI)
        apply arith
      apply(force intro:inv2_mono)
  apply clarsimp
  apply arith
done

```

```

theorem "inv ins  $\implies$  inv (del1 i ins)"
by(simp only:inv_def inv2_del1[of 0])

```

```

lemma inv2_yields_lb[rule_format]:
  "∀m. inv2 m ins → n < m → n ∉ set_of ins"
by(induct ins, auto)

lemma [simp]: "{k::nat. x <= k & k <= x} = {x}"
by(rule set_ext, simp, arith)

lemma [simp]: "n < m ⇒ {k::nat. m <= k & k <= n} = {}"
by(simp)

lemma [simp]: "0 < (n::nat) ⇒
  {k. m <= k & k <= n} - {n} = {k. m <= k & k <= n - 1}"
by(rule set_ext, simp, arith)

lemma [simp]: "{k::nat. m <= k & k <= n} - {m} = {k. Suc m <= k & k <= n}"
by(rule set_ext, simp, arith)

lemma set_of_del1[rule_format]:
  "∀m. inv2 m ins → set_of(del1 i ins) = (set_of ins) - {i}"
apply(induct ins)
  apply(simp)
  apply(clarsimp)
  apply(rule conjI)
  apply clarify
  apply(rule conjI)
  apply clarify
  apply(drule_tac n = i in inv2_yields_lb)
  apply simp
  apply blast
  apply clarify
  apply(drule_tac n = i in inv2_yields_lb)
  apply arith
  apply (simp add: Un_Diff)
  apply clarify
  apply(rule conjI)
  apply clarify
  apply(rule conjI)
  apply clarify
  apply(drule_tac n = i in inv2_yields_lb)
  apply simp
  apply blast
  apply clarify
  apply(rule conjI)

```

```

apply clarify
apply(drule_tac n = i in inv2_yields_lb)
  apply simp
  apply (simp add: Un_Diff)
apply clarify
apply(rule conjI)
  apply clarify
  apply(drule_tac n = i in inv2_yields_lb)
    apply simp
    apply (simp add: Un_Diff)
  apply clarify
  apply(drule_tac n = i in inv2_yields_lb)
    apply simp
    apply (simp add: Un_Diff)
  apply(rule set_ext)
  apply simp
  apply arith
apply clarify
apply(rule conjI)
  apply clarify
  apply(simp)
  apply blast
apply force
done

theorem "inv ins  $\implies$  set_of(del1 i ins) = (set_of ins) - {i}"
by(simp only:inv_def set_of_del1[of 0])

end

```