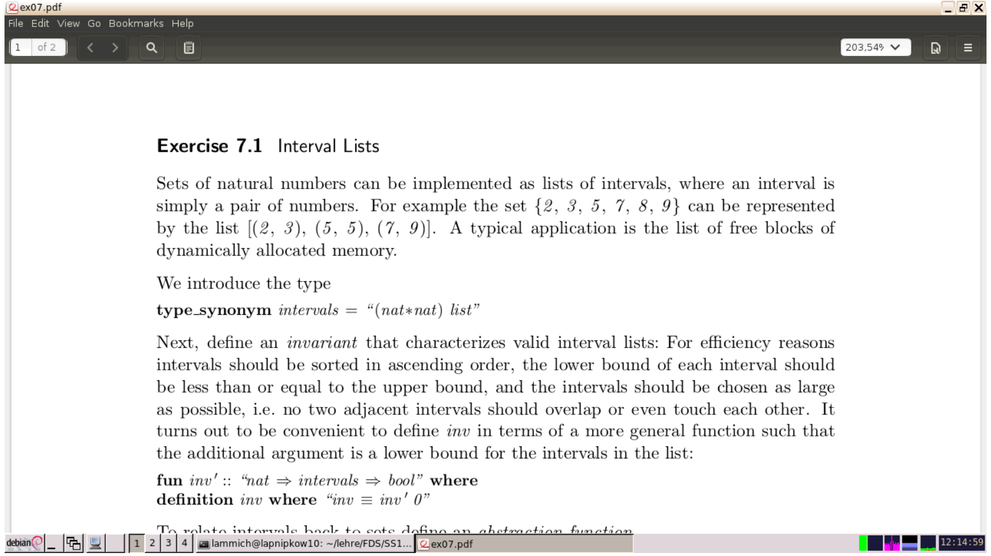
Script generated by TTT

Title: Lammich: FDS Tutorial (25.05.2018)

Date: Fri May 25 12:15:00 CEST 2018

Duration: 97:01 min

Pages: 112



ex07.pdf

Exercise 7.1 Interval Lists

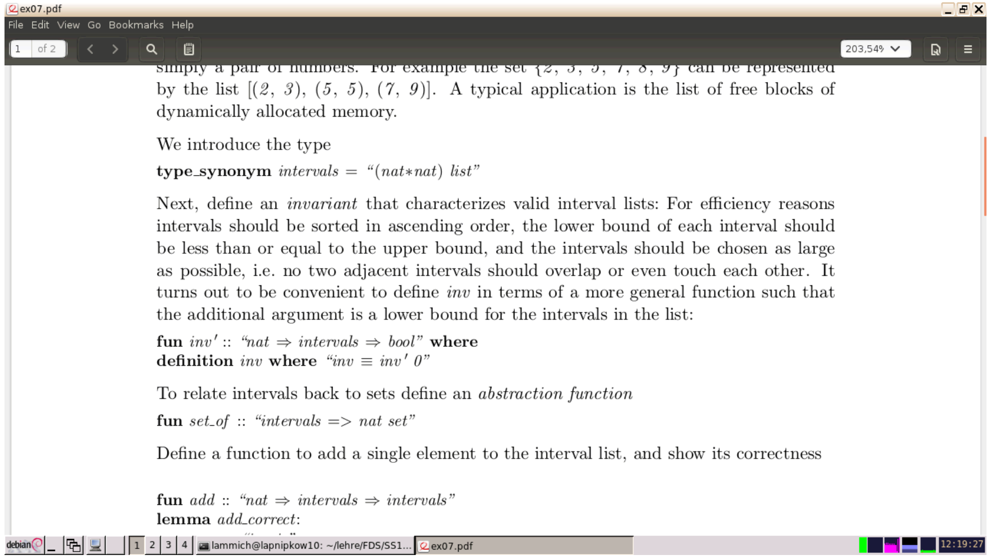
Sets of natural numbers can be implemented as lists of intervals, where an interval is simply a pair of numbers. For example the set $\{2, 3, 5, 7, 8, 9\}$ can be represented by the list $[(2, 3), (5, 5), (7, 9)]$. A typical application is the list of free blocks of dynamically allocated memory.

We introduce the type

type_synonym *intervals* = "(*nat*∗*nat*) *list*"

Next, define an *invariant* that characterizes valid interval lists: For efficiency reasons intervals should be sorted in ascending order, the lower bound of each interval should be less than or equal to the upper bound, and the intervals should be chosen as large as possible, i.e. no two adjacent intervals should overlap or even touch each other. It turns out to be convenient to define *inv* in terms of a more general function such that the additional argument is a lower bound for the intervals in the list:

fun *inv'* :: "*nat* ⇒ *intervals* ⇒ *bool*" **where**
definition *inv* **where** "*inv* ≡ *inv' 0*"



ex07.pdf

by the list $[(2, 3), (5, 5), (7, 9)]$. A typical application is the list of free blocks of dynamically allocated memory.

We introduce the type

type_synonym *intervals* = "(*nat*∗*nat*) *list*"

Next, define an *invariant* that characterizes valid interval lists: For efficiency reasons intervals should be sorted in ascending order, the lower bound of each interval should be less than or equal to the upper bound, and the intervals should be chosen as large as possible, i.e. no two adjacent intervals should overlap or even touch each other. It turns out to be convenient to define *inv* in terms of a more general function such that the additional argument is a lower bound for the intervals in the list:

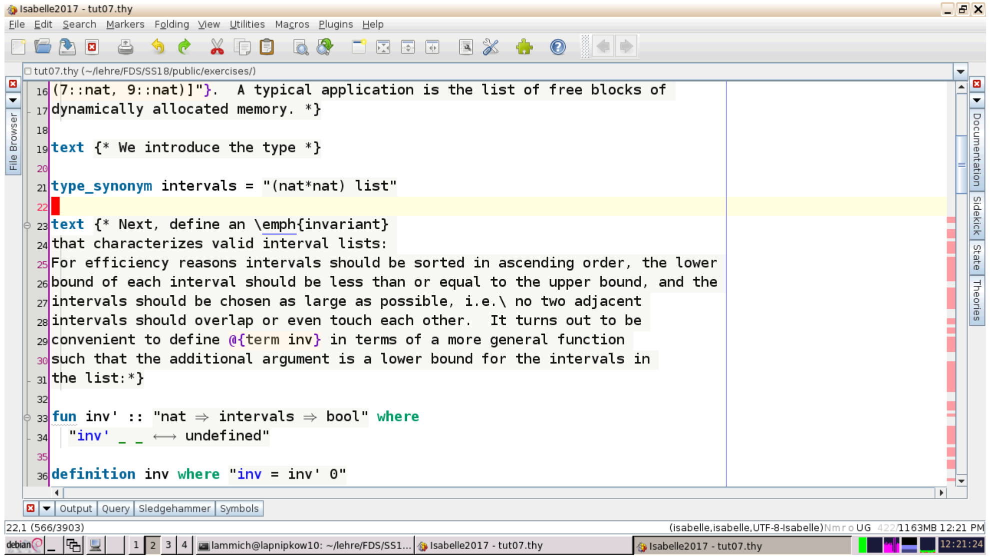
fun *inv'* :: "*nat* ⇒ *intervals* ⇒ *bool*" **where**
definition *inv* **where** "*inv* ≡ *inv' 0*"

To relate intervals back to sets define an *abstraction function*

fun *set_of* :: "*intervals* => *nat set*"

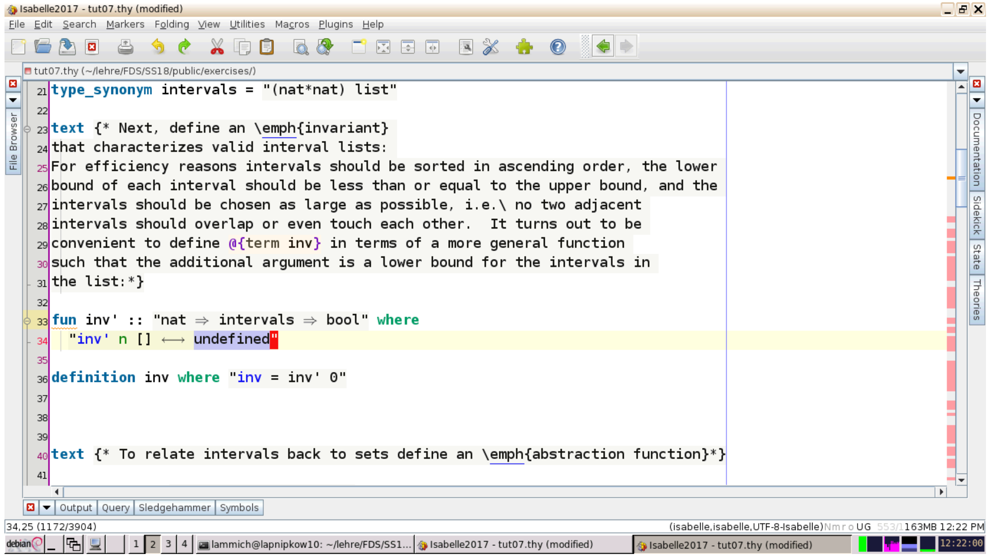
Define a function to add a single element to the interval list, and show its correctness

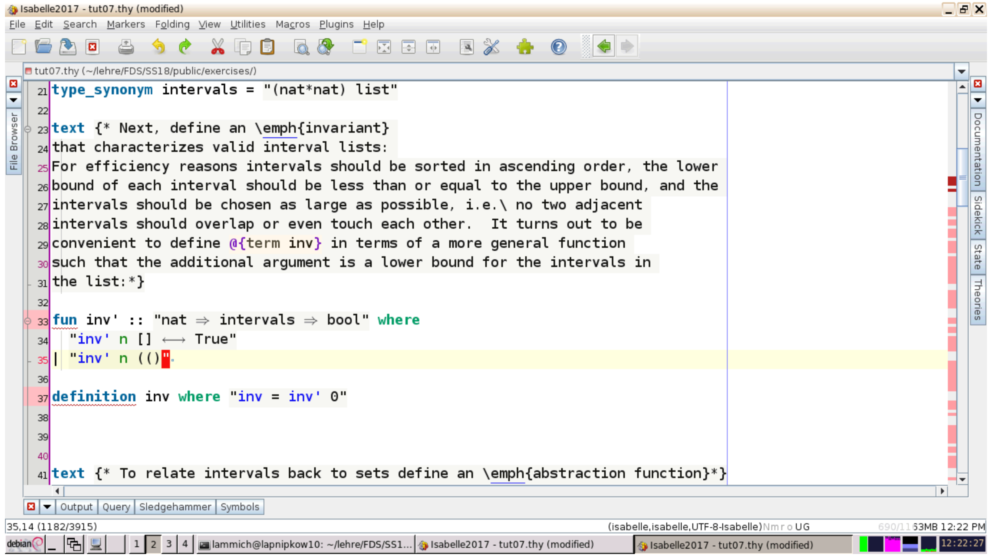
fun *add* :: "*nat* ⇒ *intervals* ⇒ *intervals*"
lemma *add_correct*:

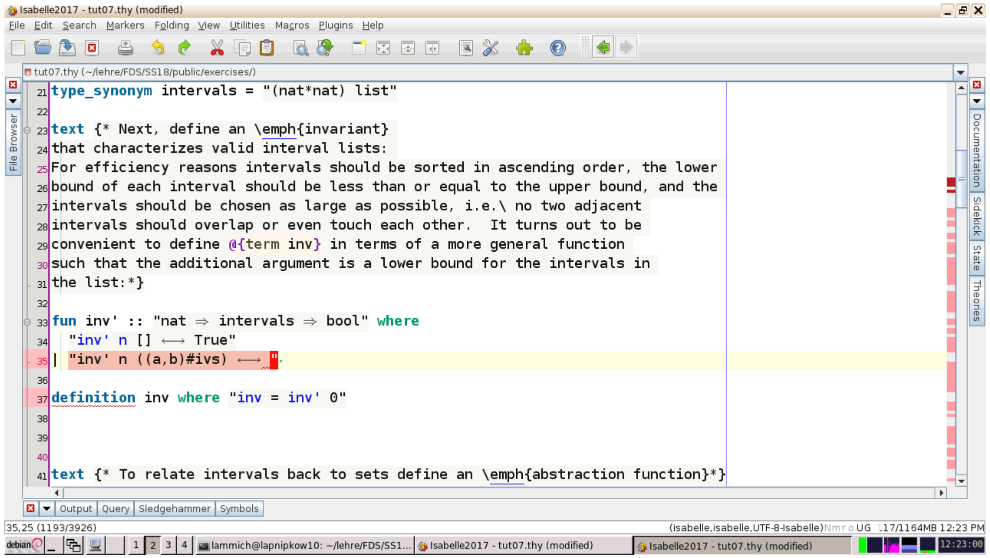


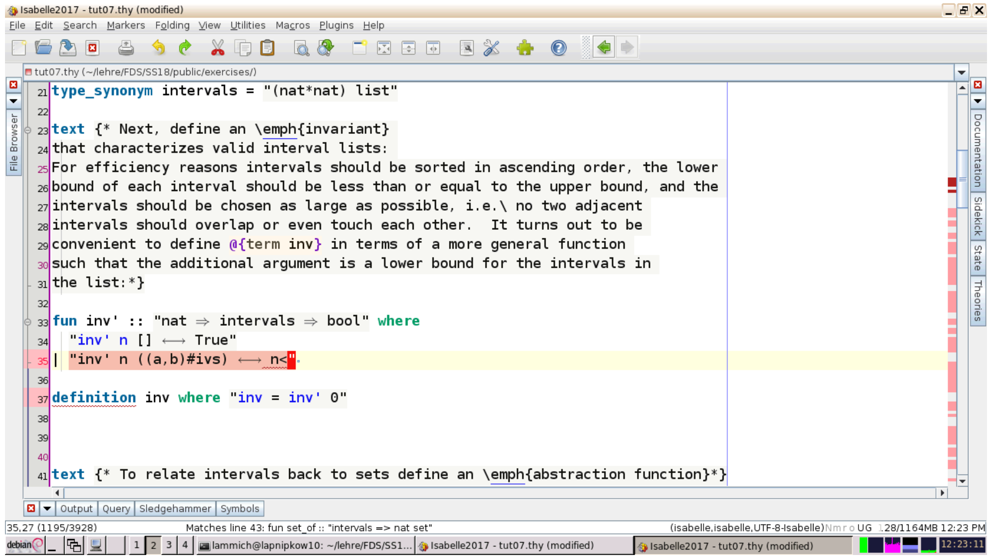
Isabelle2017 - tut07.thy

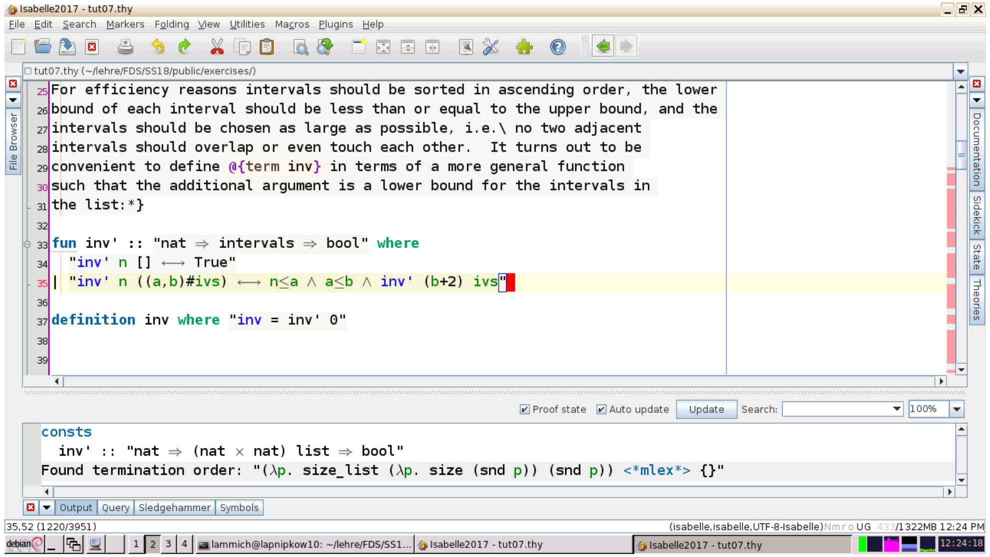
```
(7::nat, 9::nat)]"}.  A typical application is the list of free blocks of
dynamically allocated memory. *}

text {* We introduce the type *}

type_synonym intervals = "(nat*nat) list"

text {* Next, define an \emph{invariant}
that characterizes valid interval lists:
For efficiency reasons intervals should be sorted in ascending order, the lower
bound of each interval should be less than or equal to the upper bound, and the
intervals should be chosen as large as possible, i.e.\ no two adjacent
intervals should overlap or even touch each other.  It turns out to be
convenient to define @{term inv} in terms of a more general function
such that the additional argument is a lower bound for the intervals in
the list:*}

fun inv' :: "nat ⇒ intervals ⇒ bool" where
  "inv' _ _ ⟷ undefined"

definition inv where "inv = inv' 0"
```

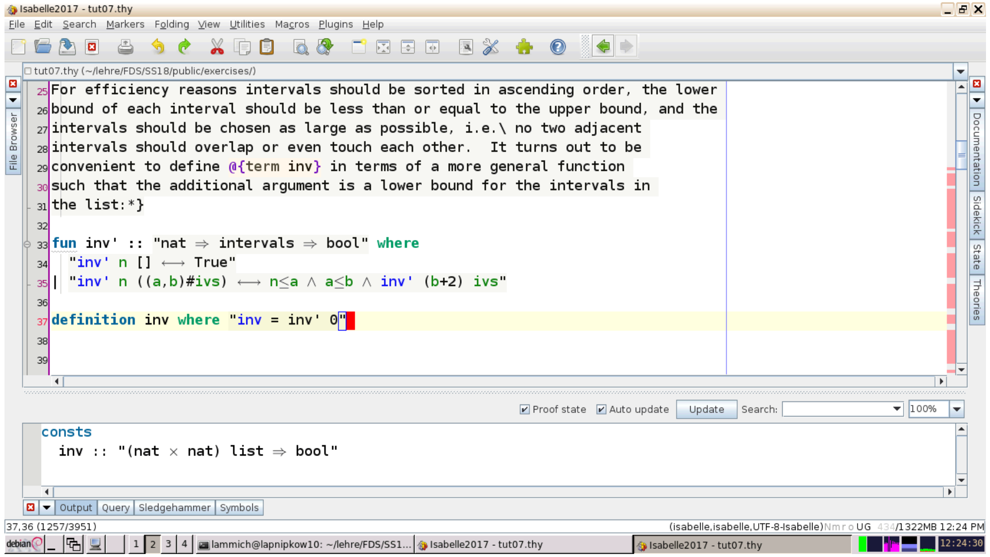

Isabelle2017 - tut07.thy (modified)
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
type_synonym intervals = "(nat*nat) list"
text {* Next, define an \emph{invariant}
that characterizes valid interval lists:
For efficiency reasons intervals should be sorted in ascending order, the lower
bound of each interval should be less than or equal to the upper bound, and the
intervals should be chosen as large as possible, i.e.\ no two adjacent
intervals should overlap or even touch each other. It turns out to be
convenient to define @{term inv} in terms of a more general function
such that the additional argument is a lower bound for the intervals in
the list:*}
fun inv' :: "nat ⇒ intervals ⇒ bool" where
"inv' n [] ⟷ undefined"
definition inv where "inv = inv' 0"
text {* To relate intervals back to sets define an \emph{abstraction function}*}
Output Query Sledgehammer Symbols
34,25 (1172/3904)

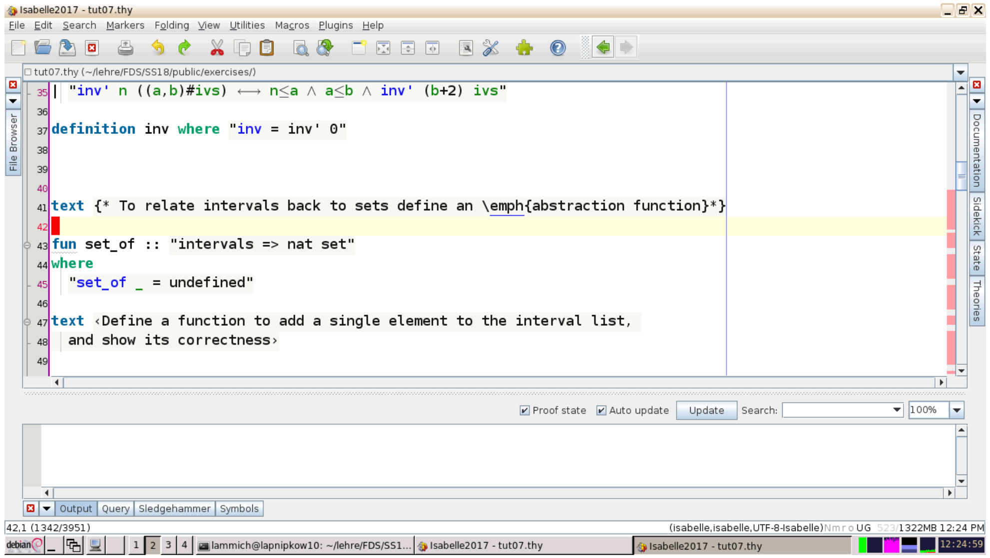

Isabelle2017 - tut07.thy (modified)
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
type_synonym intervals = "(nat*nat) list"
text {* Next, define an \emph{invariant}
that characterizes valid interval lists:
For efficiency reasons intervals should be sorted in ascending order, the lower
bound of each interval should be less than or equal to the upper bound, and the
intervals should be chosen as large as possible, i.e.\ no two adjacent
intervals should overlap or even touch each other. It turns out to be
convenient to define @{term inv} in terms of a more general function
such that the additional argument is a lower bound for the intervals in
the list:*}
fun inv' :: "nat ⇒ intervals ⇒ bool" where
"inv' n [] ⟷ True"
| "inv' n (()"
definition inv where "inv = inv' 0"
text {* To relate intervals back to sets define an \emph{abstraction function}*}
Output Query Sledgehammer Symbols
35,14 (1182/3915)

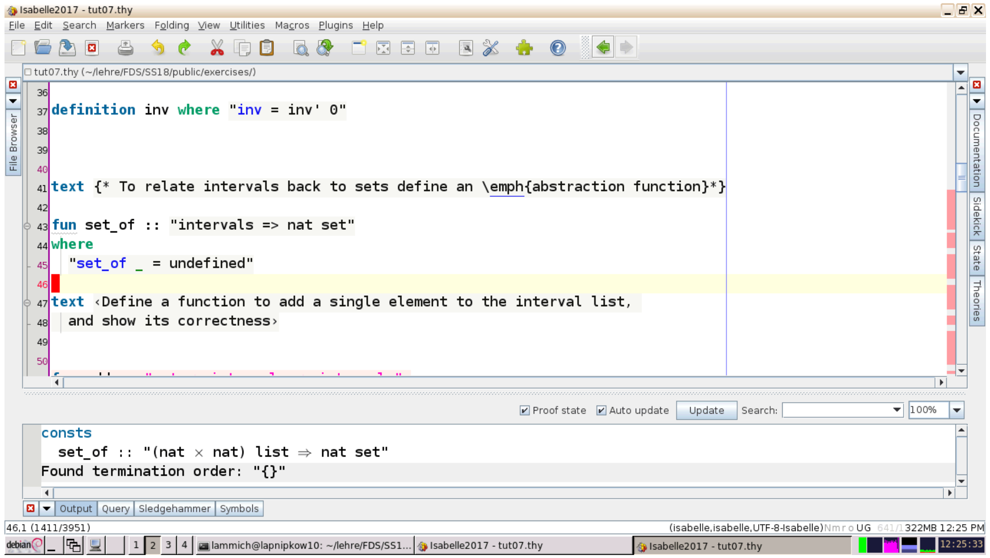

Isabelle2017 - tut07.thy (modified)
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
type_synonym intervals = "(nat*nat) list"
text {* Next, define an \emph{invariant}
that characterizes valid interval lists:
For efficiency reasons intervals should be sorted in ascending order, the lower
bound of each interval should be less than or equal to the upper bound, and the
intervals should be chosen as large as possible, i.e.\ no two adjacent
intervals should overlap or even touch each other. It turns out to be
convenient to define @{term inv} in terms of a more general function
such that the additional argument is a lower bound for the intervals in
the list:*}
fun inv' :: "nat ⇒ intervals ⇒ bool" where
"inv' n [] ⟷ True"
| "inv' n ((a,b)#ivs) ⟷ _"
definition inv where "inv = inv' 0"
text {* To relate intervals back to sets define an \emph{abstraction function}*}
Output Query Sledgehammer Symbols
35,25 (1193/3926)

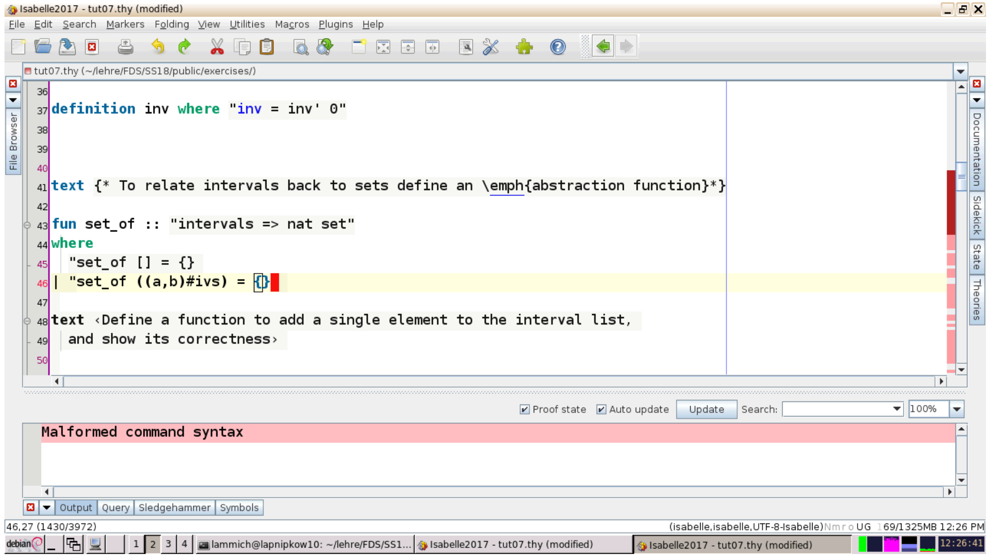

Isabelle2017 - tut07.thy (modified)
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
type_synonym intervals = "(nat*nat) list"
text {* Next, define an \emph{invariant}
that characterizes valid interval lists:
For efficiency reasons intervals should be sorted in ascending order, the lower
bound of each interval should be less than or equal to the upper bound, and the
intervals should be chosen as large as possible, i.e.\ no two adjacent
intervals should overlap or even touch each other. It turns out to be
convenient to define @{term inv} in terms of a more general function
such that the additional argument is a lower bound for the intervals in
the list:*}
fun inv' :: "nat ⇒ intervals ⇒ bool" where
"inv' n [] ⟷ True"
| "inv' n ((a,b)#ivs) ⟷ n≤"
definition inv where "inv = inv' 0"
text {* To relate intervals back to sets define an \emph{abstraction function}*}
Output Query Sledgehammer Symbols
35,27 (1195/3928) Matches line 43: fun set_of :: "intervals => nat set"

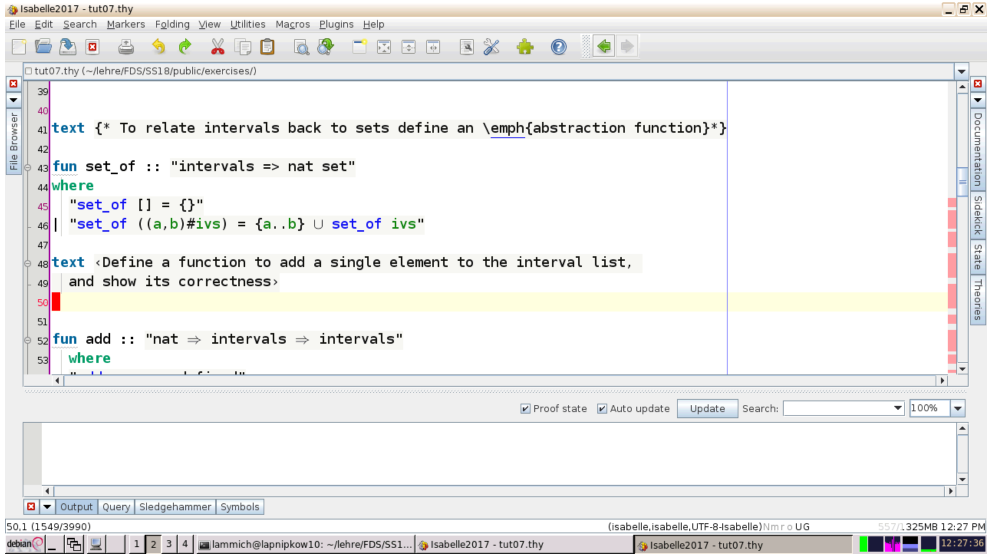

Isabelle2017 - tut07.thy
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
For efficiency reasons intervals should be sorted in ascending order, the lower
bound of each interval should be less than or equal to the upper bound, and the
intervals should be chosen as large as possible, i.e.\ no two adjacent
intervals should overlap or even touch each other. It turns out to be
convenient to define @{term inv} in terms of a more general function
such that the additional argument is a lower bound for the intervals in
the list:*}
fun inv' :: "nat ⇒ intervals ⇒ bool" where
"inv' n [] ⟷ True"
| "inv' n ((a,b)#ivs) ⟷ n≤a ∧ a≤b ∧ inv' (b+2) ivs"
definition inv where "inv = inv' 0"
consts
inv' :: "nat ⇒ (nat × nat) list ⇒ bool"
Found termination order: "(λp. size_list (λp. size (snd p)) (snd p)) <*mlex*> {}"

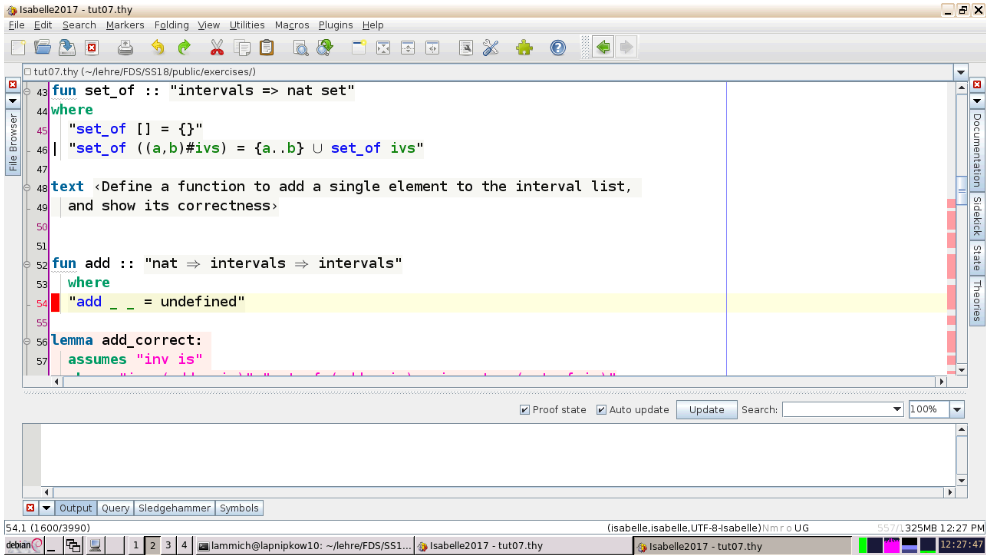

Isabelle2017 - tut07.thy
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
For efficiency reasons intervals should be sorted in ascending order, the lower
bound of each interval should be less than or equal to the upper bound, and the
intervals should be chosen as large as possible, i.e.\ no two adjacent
intervals should overlap or even touch each other. It turns out to be
convenient to define @{term inv} in terms of a more general function
such that the additional argument is a lower bound for the intervals in
the list:*}
fun inv' :: "nat ⇒ intervals ⇒ bool" where
"inv' n [] ⟷ True"
| "inv' n ((a,b)#ivs) ⟷ n≤a ∧ a≤b ∧ inv' (b+2) ivs"
definition inv where "inv = inv' 0"
consts
inv :: "(nat × nat) list ⇒ bool"

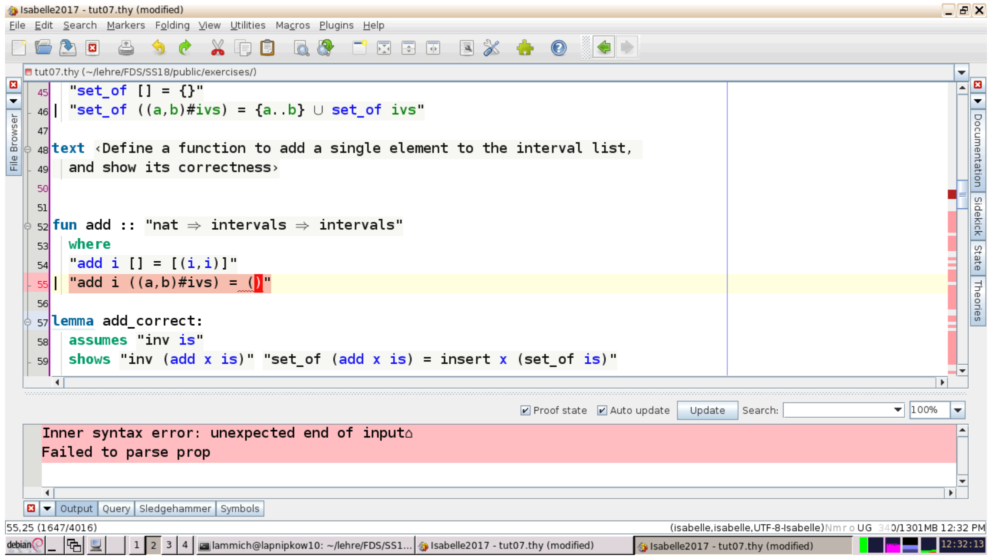

Isabelle2017 - tut07.thy
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
| "inv' n ((a,b)#ivs) ⟷ n≤a ∧ a≤b ∧ inv' (b+2) ivs"
definition inv where "inv = inv' 0"
text {* To relate intervals back to sets define an \emph{abstraction function}*}
fun set_of :: "intervals => nat set"
where
"set_of _ = undefined"
text ‹Define a function to add a single element to the interval list,
and show its correctness›

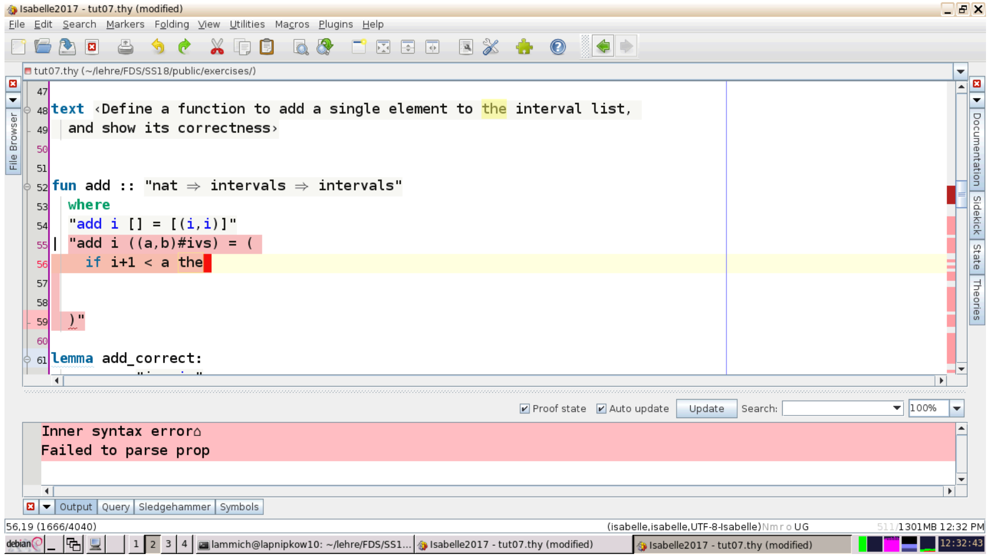

Isabelle2017 - tut07.thy
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
definition inv where "inv = inv' 0"
text {* To relate intervals back to sets define an \emph{abstraction function}*}
fun set_of :: "intervals => nat set"
where
"set_of _ = undefined"
text ‹Define a function to add a single element to the interval list,
and show its correctness›
consts
set_of :: "(nat × nat) list ⇒ nat set"
Found termination order: "{}"

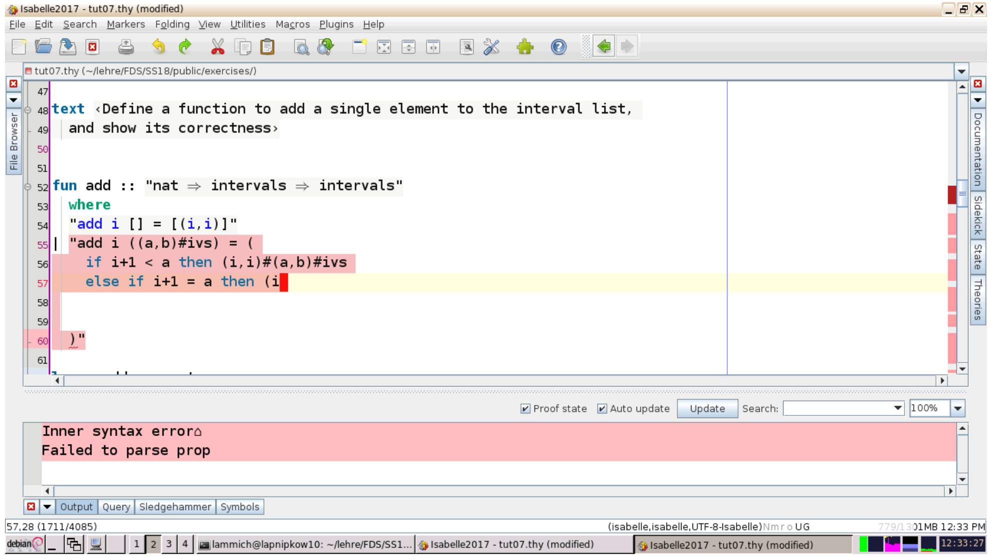

Isabelle2017 - tut07.thy (modified)
definition inv where "inv = inv' 0"
text {* To relate intervals back to sets define an \emph{abstraction function}*}
fun set_of :: "intervals => nat set"
where
"set_of [] = {}"
| "set_of ((a,b)#ivs) = {}
text ‹Define a function to add a single element to the interval list,
and show its correctness›
Malformed command syntax

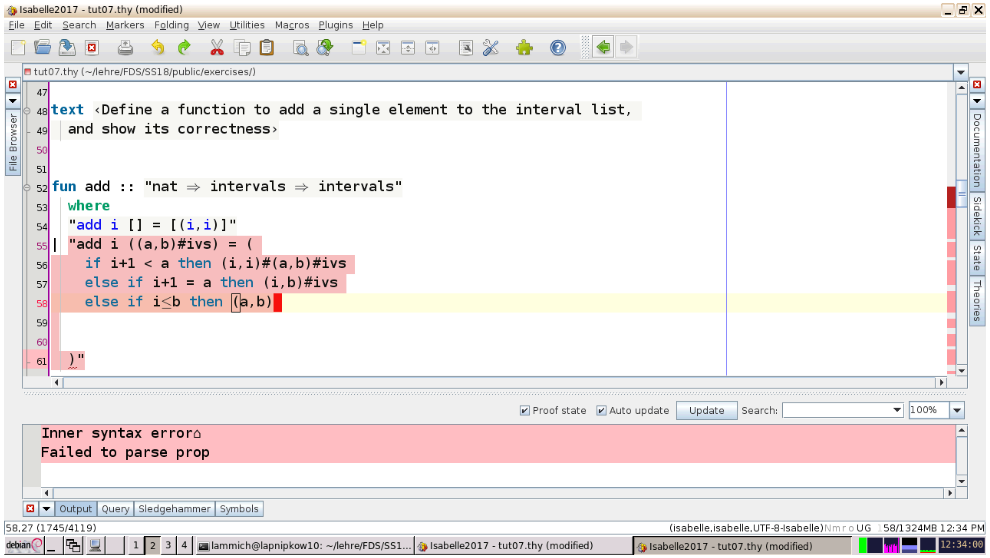

Isabelle2017 - tut07.thy
text {* To relate intervals back to sets define an \emph{abstraction function}*}
fun set_of :: "intervals => nat set"
where
"set_of [] = {}"
| "set_of ((a,b)#ivs) = {a..b} ∪ set_of ivs"
text ‹Define a function to add a single element to the interval list,
and show its correctness›
fun add :: "nat ⇒ intervals ⇒ intervals"
where

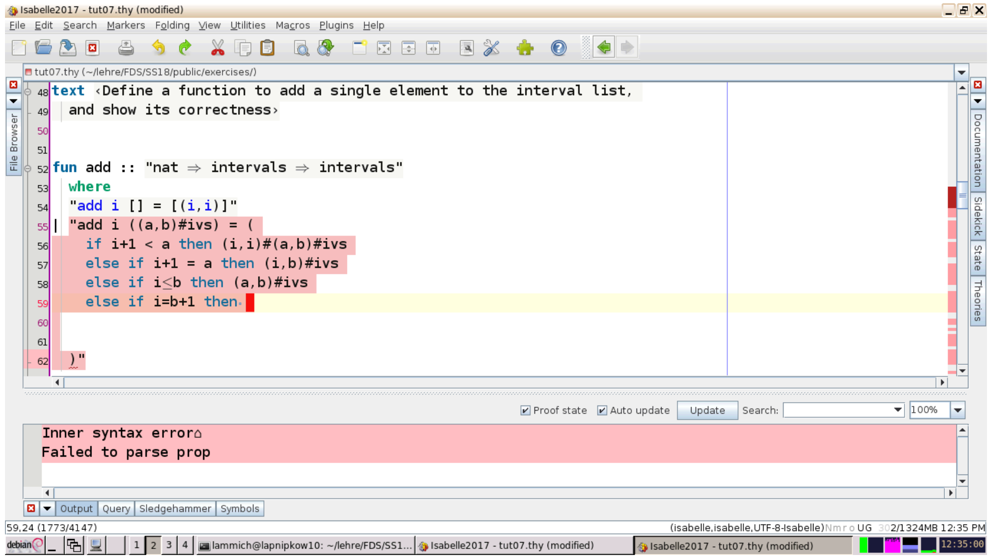

Isabelle2017 - tut07.thy
fun set_of :: "intervals => nat set"
where
"set_of [] = {}"
| "set_of ((a,b)#ivs) = {a..b} ∪ set_of ivs"
text ‹Define a function to add a single element to the interval list,
and show its correctness›
fun add :: "nat ⇒ intervals ⇒ intervals"
where
"add _ _ = undefined"
lemma add_correct:
assumes "inv is"

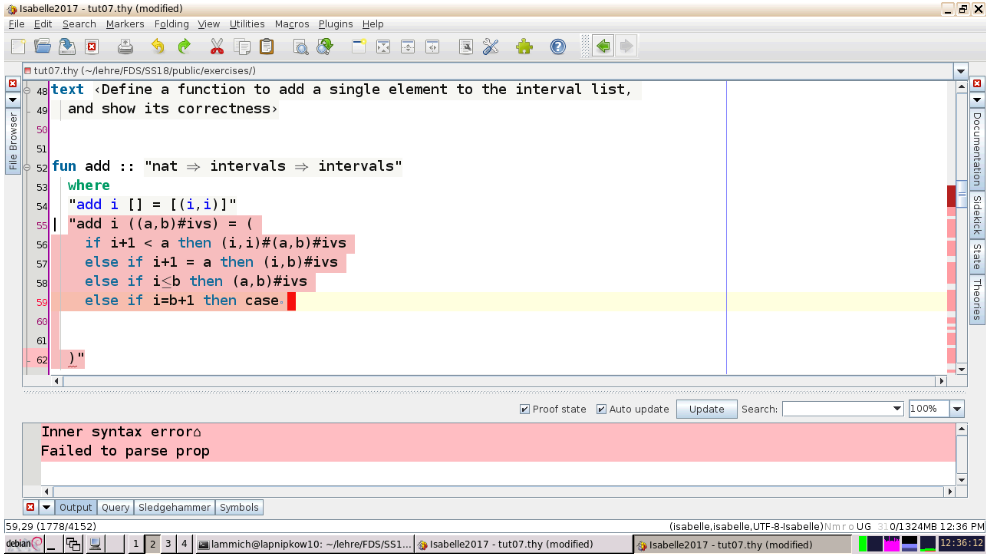

Isabelle2017 - tut07.thy (modified)
"set_of [] = {}"
| "set_of ((a,b)#ivs) = {a..b} ∪ set_of ivs"
text ‹Define a function to add a single element to the interval list,
and show its correctness›
fun add :: "nat ⇒ intervals ⇒ intervals"
where
"add i [] = [(i,i)]"
| "add i ((a,b)#ivs) = ()"
lemma add_correct:
assumes "inv is"
shows "inv (add x is)" "set_of (add x is) = insert x (set_of is)"
Inner syntax error: unexpected end of input
Failed to parse prop

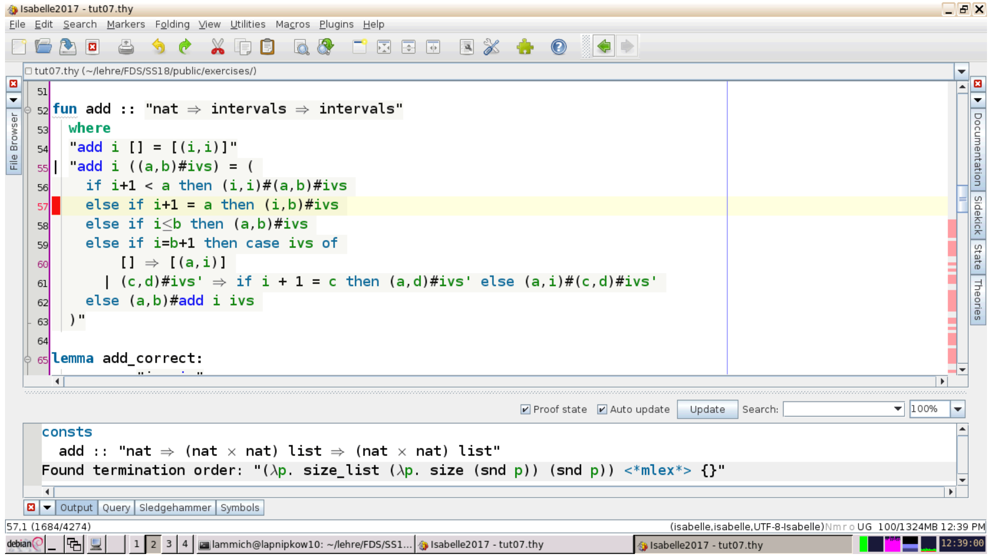

Isabelle2017 - tut07.thy (modified)
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
text ‹Define a function to add a single element to the interval list,
and show its correctness›
fun add :: "nat ⇒ intervals ⇒ intervals"
where
"add i [] = [(i,i)]"
| "add i ((a,b)#ivs) = (
if i+1 < a the
)"
lemma add_correct:
Inner syntax error
Failed to parse prop

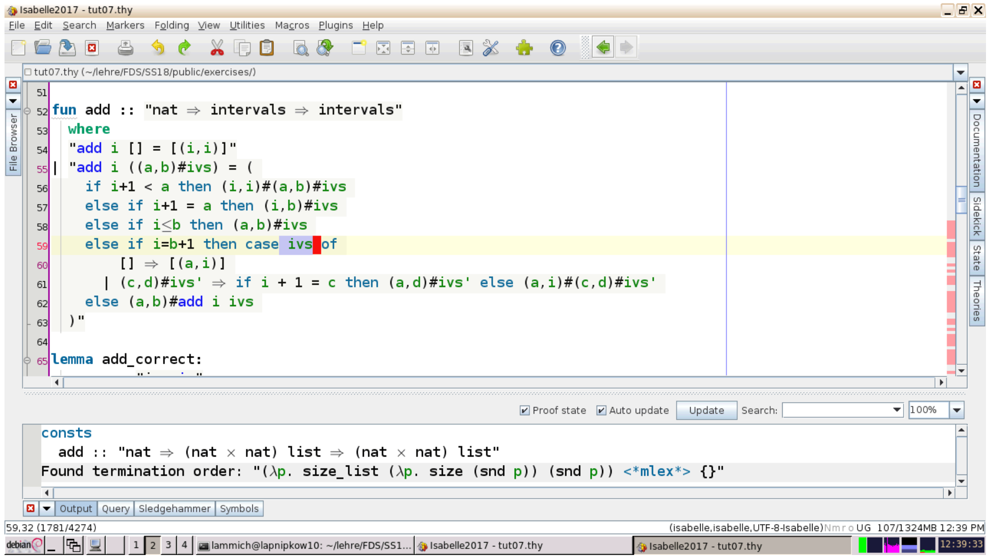

Isabelle2017 - tut07.thy (modified)
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
text ‹Define a function to add a single element to the interval list,
and show its correctness›
fun add :: "nat ⇒ intervals ⇒ intervals"
where
"add i [] = [(i,i)]"
| "add i ((a,b)#ivs) = (
if i+1 < a then (i,i)#(a,b)#ivs
else if i+1 = a then (i
)"
Inner syntax error
Failed to parse prop

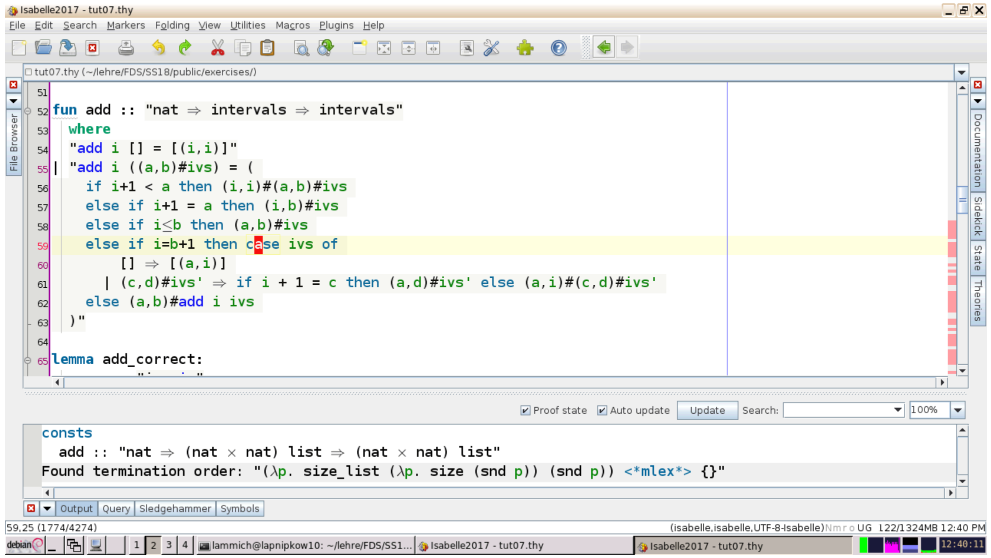

Isabelle2017 - tut07.thy (modified)
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
text ‹Define a function to add a single element to the interval list,
and show its correctness›
fun add :: "nat ⇒ intervals ⇒ intervals"
where
"add i [] = [(i,i)]"
| "add i ((a,b)#ivs) = (
if i+1 < a then (i,i)#(a,b)#ivs
else if i+1 = a then (i,b)#ivs
else if i≤b then (a,b)
)"
Inner syntax error
Failed to parse prop

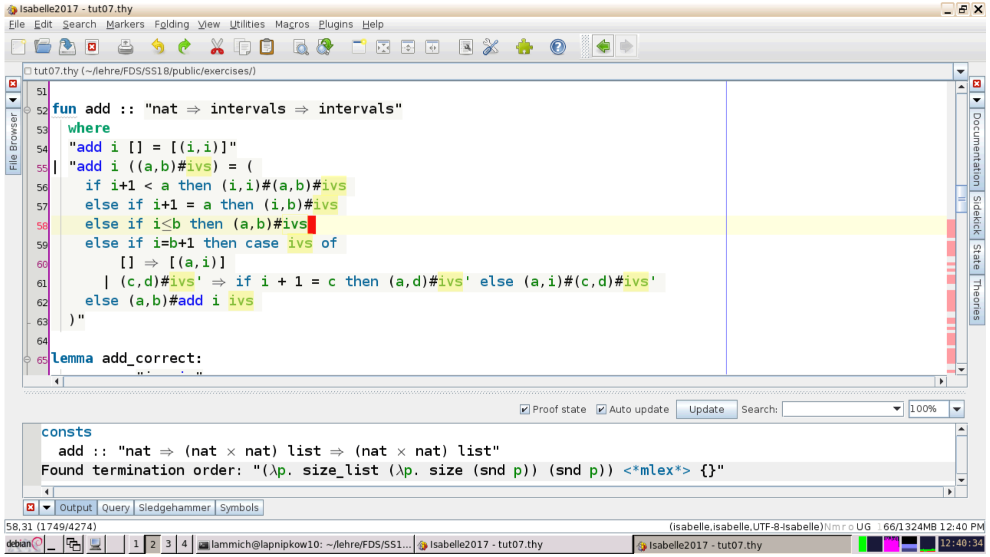
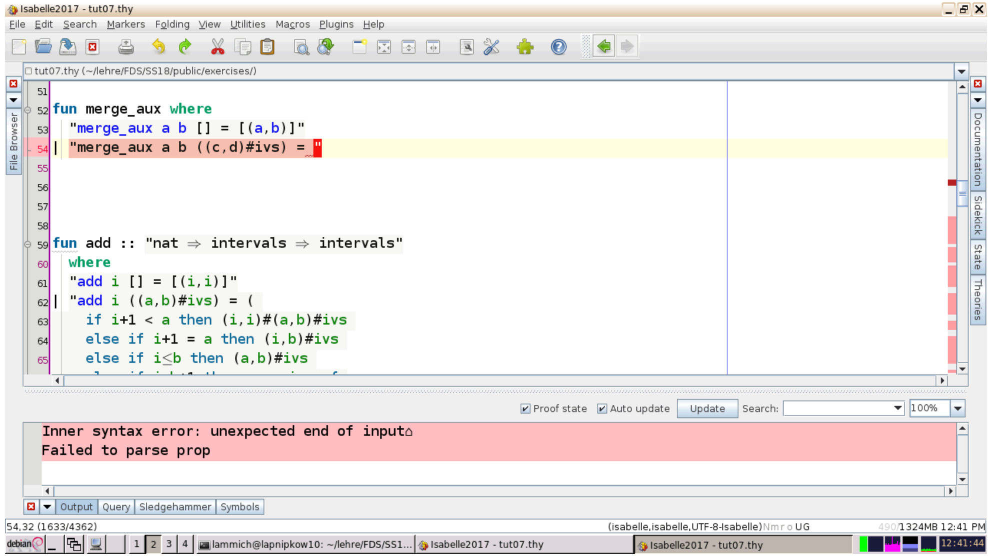
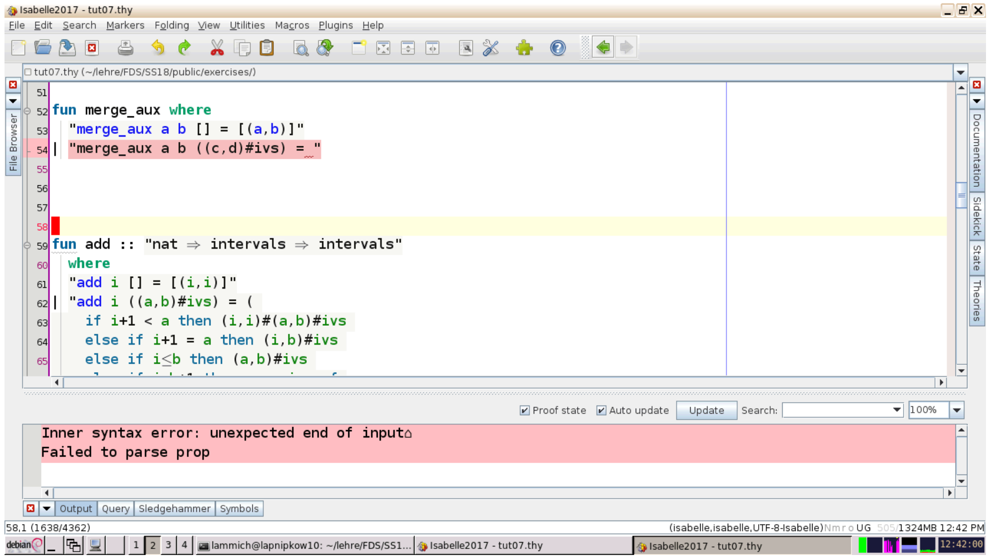
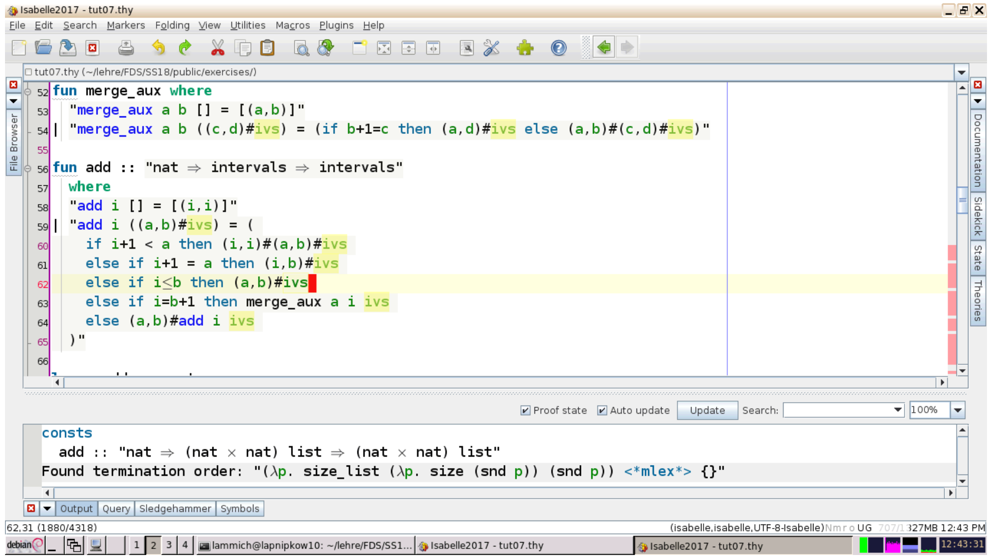

Isabelle2017 - tut07.thy (modified)
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
text ‹Define a function to add a single element to the interval list,
and show its correctness›
fun add :: "nat ⇒ intervals ⇒ intervals"
where
"add i [] = [(i,i)]"
| "add i ((a,b)#ivs) = (
if i+1 < a then (i,i)#(a,b)#ivs
else if i+1 = a then (i,b)#ivs
else if i≤b then (a,b)#ivs
else if i=b+1 then
)"
Inner syntax error
Failed to parse prop

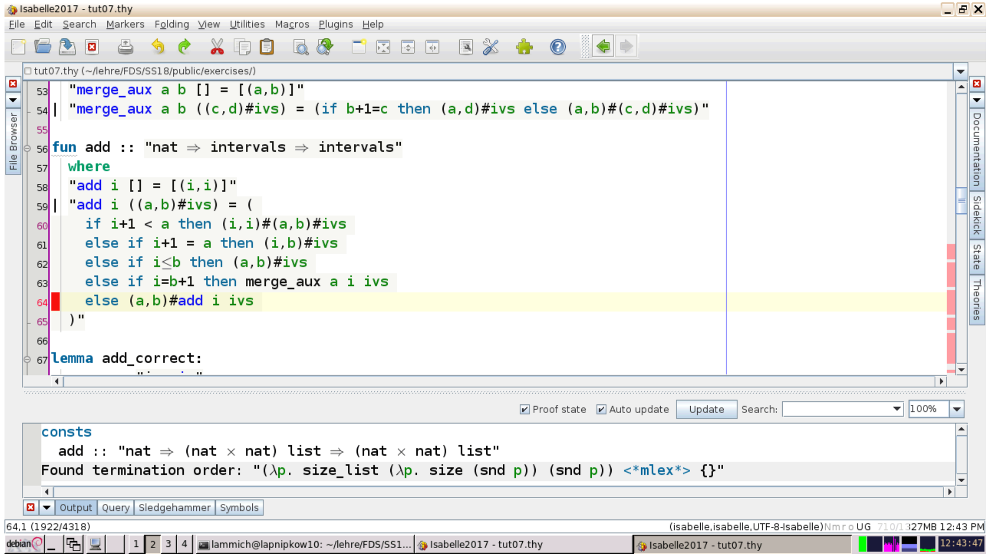
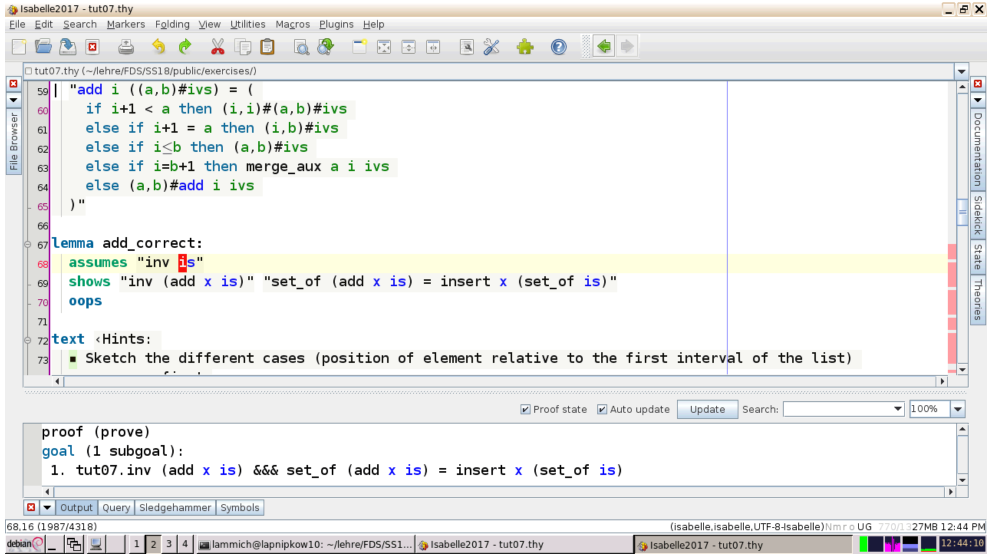
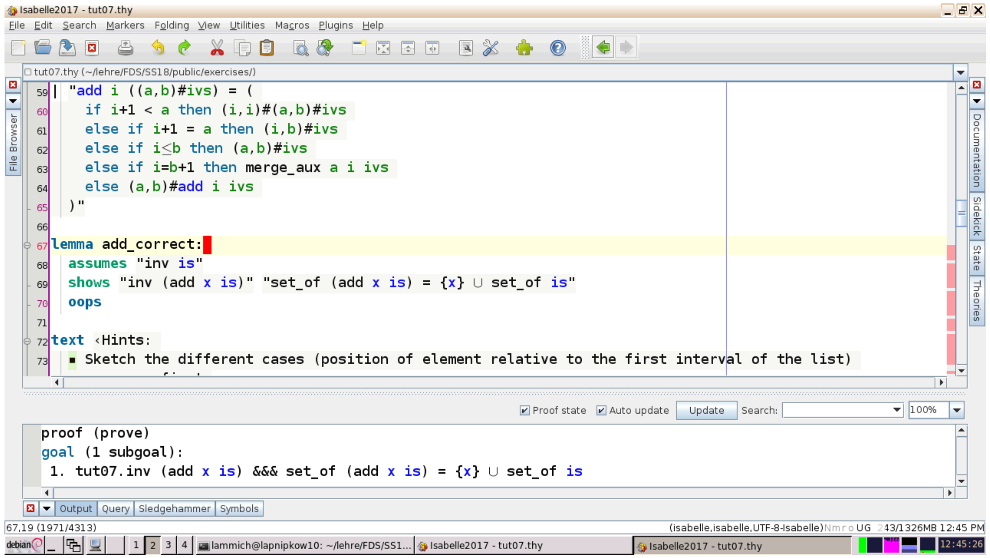
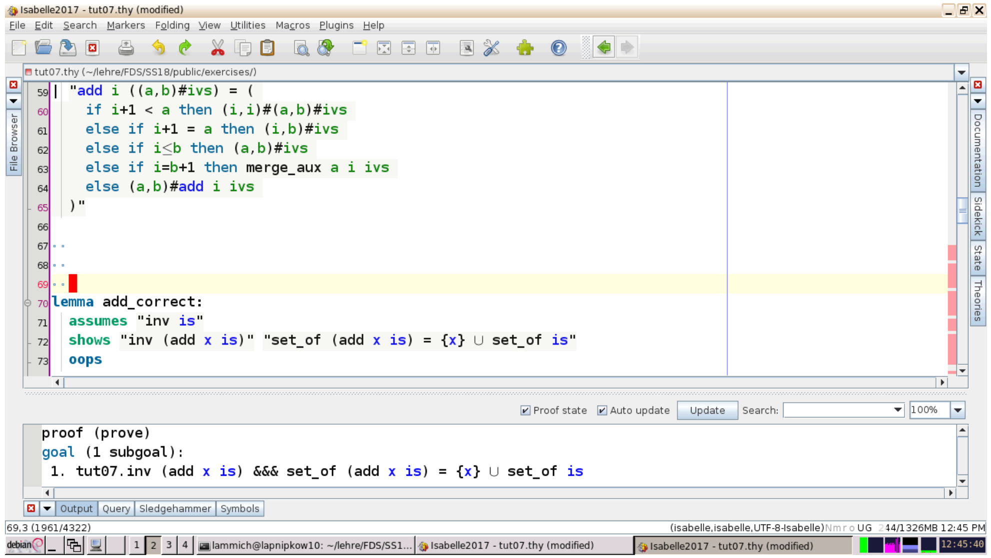

Isabelle2017 - tut07.thy (modified)
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
text ‹Define a function to add a single element to the interval list,
and show its correctness›
fun add :: "nat ⇒ intervals ⇒ intervals"
where
"add i [] = [(i,i)]"
| "add i ((a,b)#ivs) = (
if i+1 < a then (i,i)#(a,b)#ivs
else if i+1 = a then (i,b)#ivs
else if i≤b then (a,b)#ivs
else if i=b+1 then case
)"
Inner syntax error
Failed to parse prop

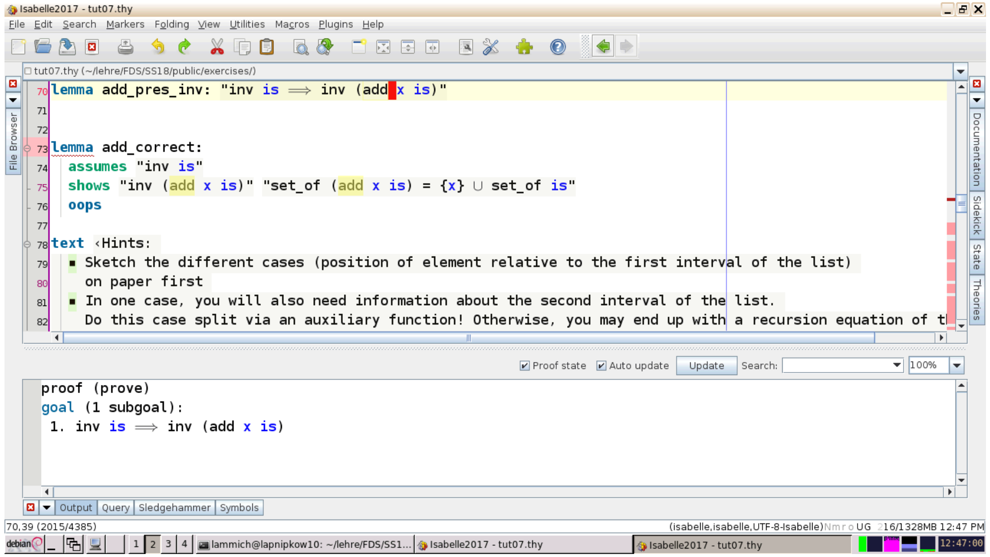

Isabelle2017 - tut07.thy
fun add :: "nat ⇒ intervals ⇒ intervals"
where
"add i [] = [(i,i)]"
| "add i ((a,b)#ivs) = (
if i+1 < a then (i,i)#(a,b)#ivs
else if i+1 = a then (i,b)#ivs
else if i≤b then (a,b)#ivs
else if i=b+1 then case ivs of
[] ⇒ [(a,i)]
| (c,d)#ivs' ⇒ if i + 1 = c then (a,d)#ivs' else (a,i)#(c,d)#ivs'
else (a,b)#add i ivs
)"
lemma add_correct:
consts
add :: "nat ⇒ (nat × nat) list ⇒ (nat × nat) list"
Found termination order: "(λp. size_list (λp. size (snd p)) (snd p)) <*mlex*> {}"

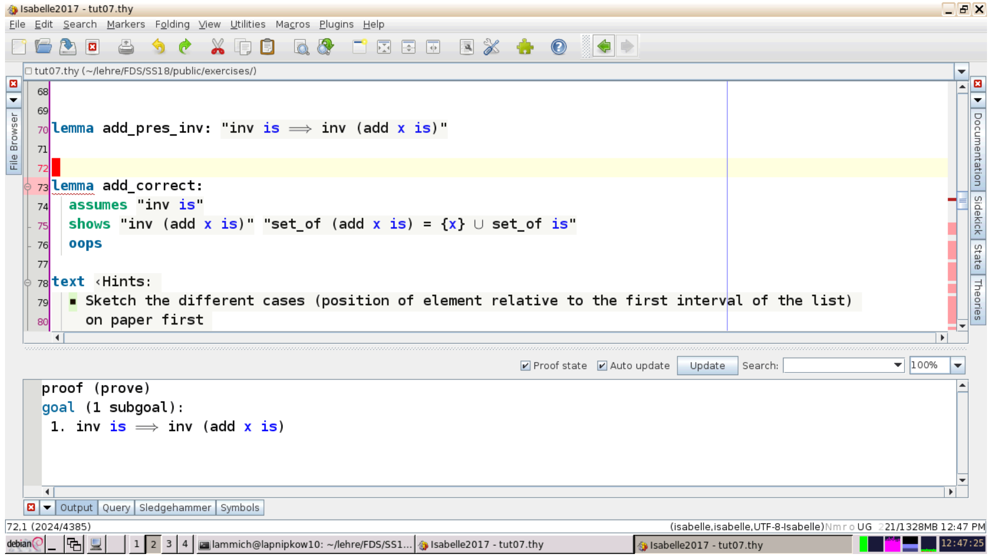

Isabelle2017 - tut07.thy
fun add :: "nat ⇒ intervals ⇒ intervals"
where
"add i [] = [(i,i)]"
| "add i ((a,b)#ivs) = (
if i+1 < a then (i,i)#(a,b)#ivs
else if i+1 = a then (i,b)#ivs
else if i≤b then (a,b)#ivs
else if i=b+1 then case ivs of
[] ⇒ [(a,i)]
| (c,d)#ivs' ⇒ if i + 1 = c then (a,d)#ivs' else (a,i)#(c,d)#ivs'
else (a,b)#add i ivs
)"
lemma add_correct:
consts
add :: "nat ⇒ (nat × nat) list ⇒ (nat × nat) list"
Found termination order: "(λp. size_list (λp. size (snd p)) (snd p)) <*mlex*> {}"

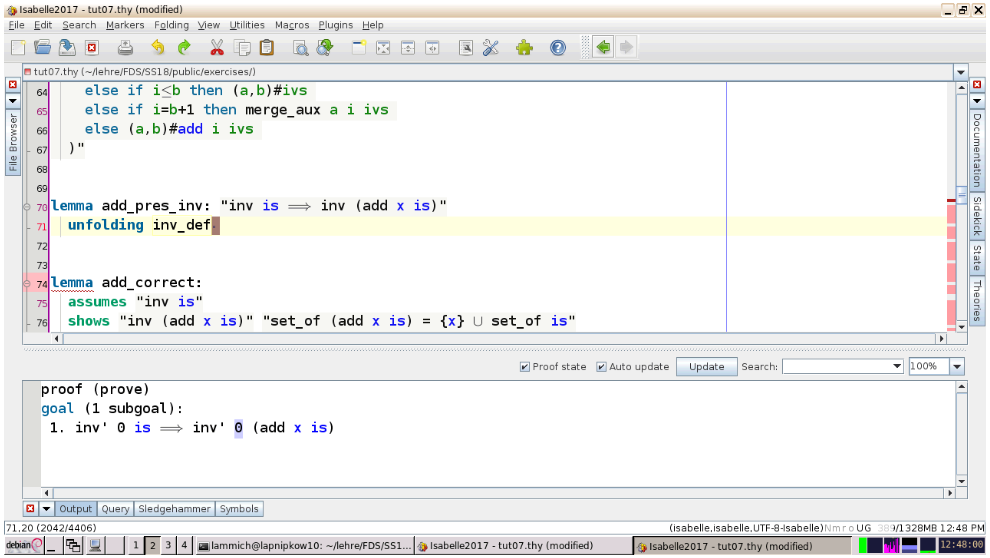

Isabelle2017 - tut07.thy
fun add :: "nat ⇒ intervals ⇒ intervals"
where
"add i [] = [(i,i)]"
| "add i ((a,b)#ivs) = (
if i+1 < a then (i,i)#(a,b)#ivs
else if i+1 = a then (i,b)#ivs
else if i≤b then (a,b)#ivs
else if i=b+1 then case ivs of
[] ⇒ [(a,i)]
| (c,d)#ivs' ⇒ if i + 1 = c then (a,d)#ivs' else (a,i)#(c,d)#ivs'
else (a,b)#add i ivs
)"
lemma add_correct:
consts
add :: "nat ⇒ (nat × nat) list ⇒ (nat × nat) list"
Found termination order: "(λp. size_list (λp. size (snd p)) (snd p)) <*mlex*> {}"

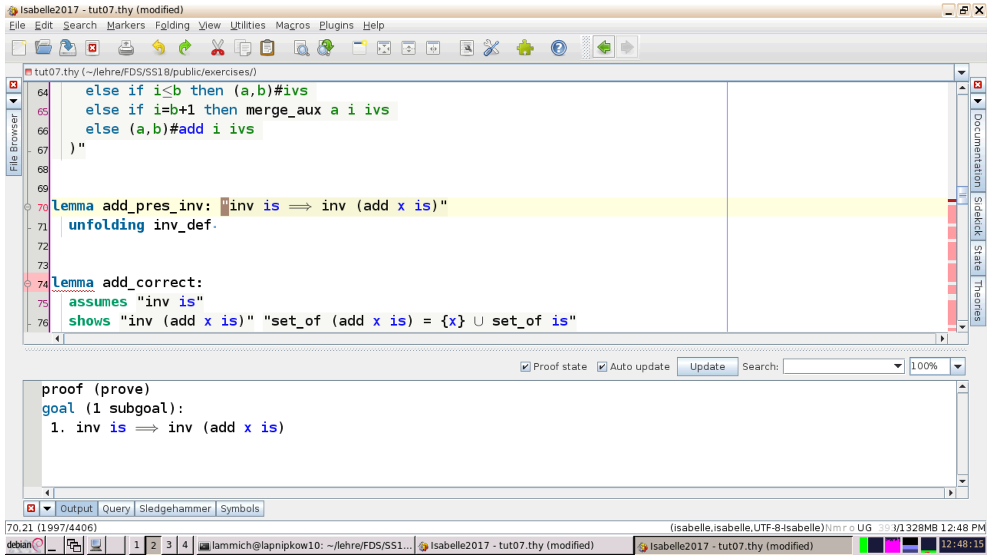

Isabelle2017 - tut07.thy
fun add :: "nat ⇒ intervals ⇒ intervals"
where
"add i [] = [(i,i)]"
| "add i ((a,b)#ivs) = (
if i+1 < a then (i,i)#(a,b)#ivs
else if i+1 = a then (i,b)#ivs
else if i≤b then (a,b)#ivs
else if i=b+1 then case ivs of
[] ⇒ [(a,i)]
| (c,d)#ivs' ⇒ if i + 1 = c then (a,d)#ivs' else (a,i)#(c,d)#ivs'
else (a,b)#add i ivs
)"
lemma add_correct:
consts
add :: "nat ⇒ (nat × nat) list ⇒ (nat × nat) list"
Found termination order: "(λp. size_list (λp. size (snd p)) (snd p)) <*mlex*> {}"

Isabelle2017 - tut07.thy
fun merge_aux where
"merge_aux a b [] = [(a,b)]"
| "merge_aux a b ((c,d)#ivs) = "
fun add :: "nat ⇒ intervals ⇒ intervals"
where
"add i [] = [(i,i)]"
| "add i ((a,b)#ivs) = (
if i+1 < a then (i,i)#(a,b)#ivs
else if i+1 = a then (i,b)#ivs
else if i≤b then (a,b)#ivs
Inner syntax error: unexpected end of input
Failed to parse prop

Isabelle2017 - tut07.thy
fun merge_aux where
"merge_aux a b [] = [(a,b)]"
| "merge_aux a b ((c,d)#ivs) = "
fun add :: "nat ⇒ intervals ⇒ intervals"
where
"add i [] = [(i,i)]"
| "add i ((a,b)#ivs) = (
if i+1 < a then (i,i)#(a,b)#ivs
else if i+1 = a then (i,b)#ivs
else if i≤b then (a,b)#ivs
Inner syntax error: unexpected end of input
Failed to parse prop

Isabelle2017 - tut07.thy
fun merge_aux where
"merge_aux a b [] = [(a,b)]"
| "merge_aux a b ((c,d)#ivs) = (if b+1=c then (a,d)#ivs else (a,b)#(c,d)#ivs)"
fun add :: "nat ⇒ intervals ⇒ intervals"
where
"add i [] = [(i,i)]"
| "add i ((a,b)#ivs) = (
if i+1 < a then (i,i)#(a,b)#ivs
else if i+1 = a then (i,b)#ivs
else if i≤b then (a,b)#ivs
else if i=b+1 then merge_aux a i ivs
else (a,b)#add i ivs
)"
consts
add :: "nat ⇒ (nat × nat) list ⇒ (nat × nat) list"
Found termination order: "(λp. size_list (λp. size (snd p)) (snd p)) <*mlex*> {}"

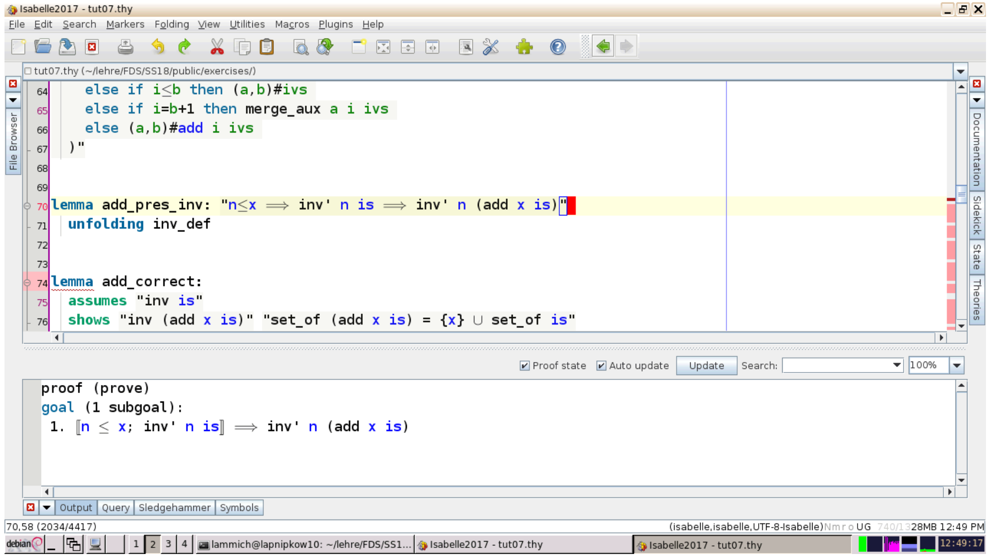

Isabelle2017 - tut07.thy
"merge_aux a b [] = [(a,b)]"
| "merge_aux a b ((c,d)#ivs) = (if b+1=c then (a,d)#ivs else (a,b)#(c,d)#ivs)"
fun add :: "nat ⇒ intervals ⇒ intervals"
where
"add i [] = [(i,i)]"
| "add i ((a,b)#ivs) = (
if i+1 < a then (i,i)#(a,b)#ivs
else if i+1 = a then (i,b)#ivs
else if i≤b then (a,b)#ivs
else if i=b+1 then merge_aux a i ivs
else (a,b)#add i ivs
)"
lemma add_correct:
consts
add :: "nat ⇒ (nat × nat) list ⇒ (nat × nat) list"
Found termination order: "(λp. size_list (λp. size (snd p)) (snd p)) <*mlex*> {}"

Isabelle2017 - tut07.thy
| "add i ((a,b)#ivs) = (
if i+1 < a then (i,i)#(a,b)#ivs
else if i+1 = a then (i,b)#ivs
else if i≤b then (a,b)#ivs
else if i=b+1 then merge_aux a i ivs
else (a,b)#add i ivs
)"
lemma add_correct:
assumes "inv is"
shows "inv (add x is)" "set_of (add x is) = insert x (set_of is)"
oops
text ‹Hints:
▪ Sketch the different cases (position of element relative to the first interval of the list)
proof (prove)
goal (1 subgoal):
1. tut07.inv (add x is) &&& set_of (add x is) = insert x (set_of is)

Isabelle2017 - tut07.thy
| "add i ((a,b)#ivs) = (
if i+1 < a then (i,i)#(a,b)#ivs
else if i+1 = a then (i,b)#ivs
else if i≤b then (a,b)#ivs
else if i=b+1 then merge_aux a i ivs
else (a,b)#add i ivs
)"
lemma add_correct:
assumes "inv is"
shows "inv (add x is)" "set_of (add x is) = {x} ∪ set_of is"
oops
text ‹Hints:
▪ Sketch the different cases (position of element relative to the first interval of the list)
proof (prove)
goal (1 subgoal):
1. tut07.inv (add x is) &&& set_of (add x is) = {x} ∪ set_of is

Isabelle2017 - tut07.thy (modified)
| "add i ((a,b)#ivs) = (
if i+1 < a then (i,i)#(a,b)#ivs
else if i+1 = a then (i,b)#ivs
else if i≤b then (a,b)#ivs
else if i=b+1 then merge_aux a i ivs
else (a,b)#add i ivs
)"
lemma add_correct:
assumes "inv is"
shows "inv (add x is)" "set_of (add x is) = {x} ∪ set_of is"
oops
proof (prove)
goal (1 subgoal):
1. tut07.inv (add x is) &&& set_of (add x is) = {x} ∪ set_of is

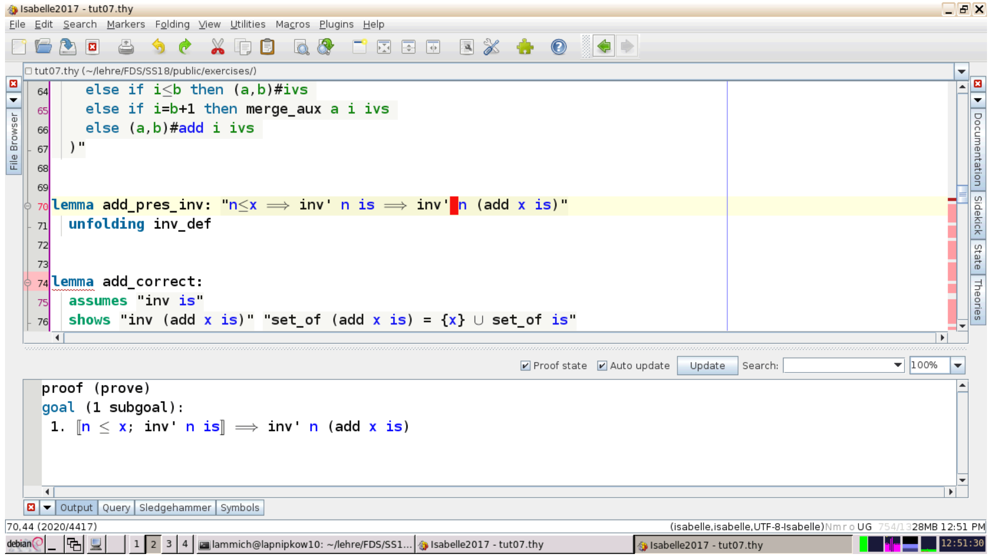

Isabelle2017 - tut07.thy
lemma add_pres_inv: "inv is ⟹ inv (add x is)"
lemma add_correct:
assumes "inv is"
shows "inv (add x is)" "set_of (add x is) = {x} ∪ set_of is"
oops
text ‹Hints:
▪ Sketch the different cases (position of element relative to the first interval of the list)
on paper first
▪ In one case, you will also need information about the second interval of the list.
Do this case split via an auxiliary function! Otherwise, you may end up with a recursion equation of t
proof (prove)
goal (1 subgoal):
1. inv is ⟹ inv (add x is)

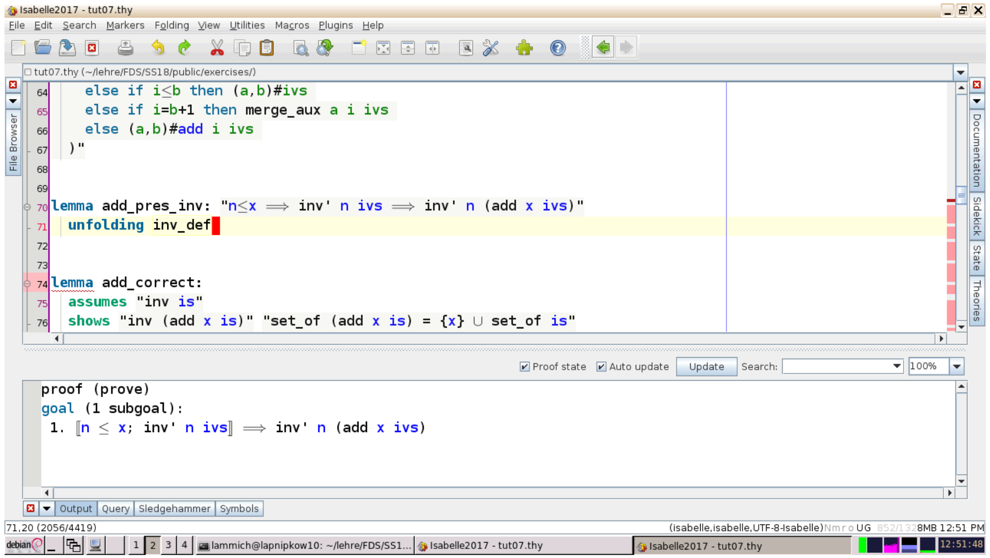

Isabelle2017 - tut07.thy
lemma add_pres_inv: "inv is ⟹ inv (add x is)"
lemma add_correct:
assumes "inv is"
shows "inv (add x is)" "set_of (add x is) = {x} ∪ set_of is"
oops
text ‹Hints:
▪ Sketch the different cases (position of element relative to the first interval of the list)
on paper first
proof (prove)
goal (1 subgoal):
1. inv is ⟹ inv (add x is)

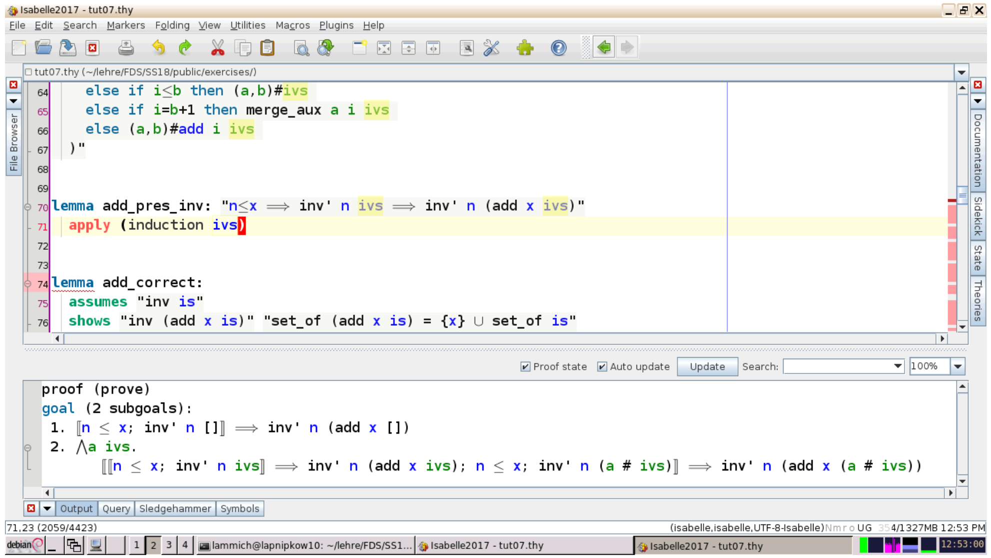

Isabelle2017 - tut07.thy (modified)
else if i≤b then (a,b)#ivs
else if i=b+1 then merge_aux a i ivs
else (a,b)#add i ivs
)"
lemma add_pres_inv: "inv is ⟹ inv (add x is)"
unfolding inv_def
lemma add_correct:
assumes "inv is"
shows "inv (add x is)" "set_of (add x is) = {x} ∪ set_of is"
proof (prove)
goal (1 subgoal):
1. inv' 0 is ⟹ inv' 0 (add x is)

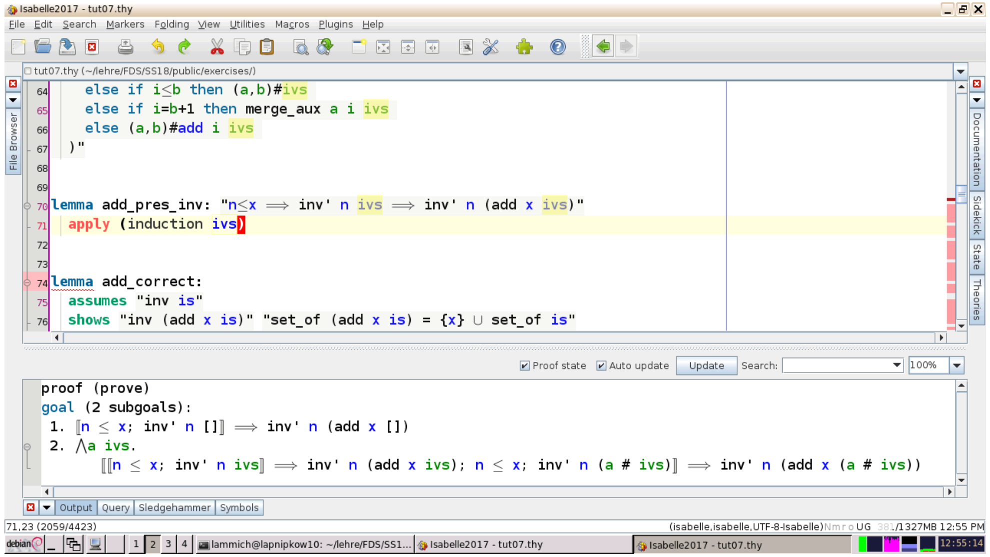

Isabelle2017 - tut07.thy (modified)
else if i≤b then (a,b)#ivs
else if i=b+1 then merge_aux a i ivs
else (a,b)#add i ivs
)"
lemma add_pres_inv: "inv is ⟹ inv (add x is)"
unfolding inv_def
lemma add_correct:
assumes "inv is"
shows "inv (add x is)" "set_of (add x is) = {x} ∪ set_of is"
proof (prove)
goal (1 subgoal):
1. inv is ⟹ inv (add x is)

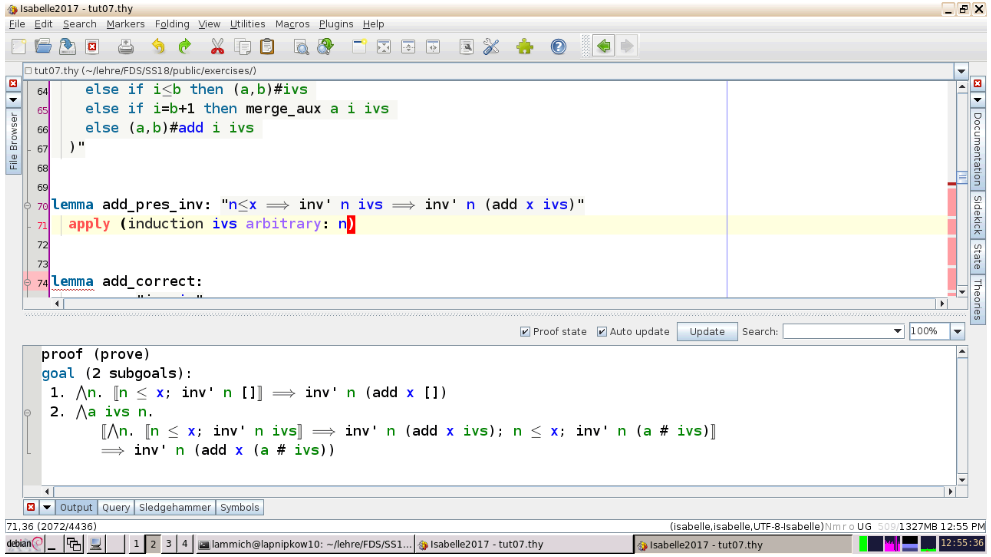

Isabelle2017 - tut07.thy
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
else if i≤b then (a,b)#ivs
else if i=b+1 then merge_aux a i ivs
else (a,b)#add i ivs
)"
lemma add_pres_inv: "n≤x ⟹ inv' n is ⟹ inv' n (add x is)"
unfolding inv_def
lemma add_correct:
assumes "inv is"
shows "inv (add x is)" "set_of (add x is) = {x} ∪ set_of is"
proof (prove)
goal (1 subgoal):
1. ⟦n ≤ x; inv' n is⟧ ⟹ inv' n (add x is)

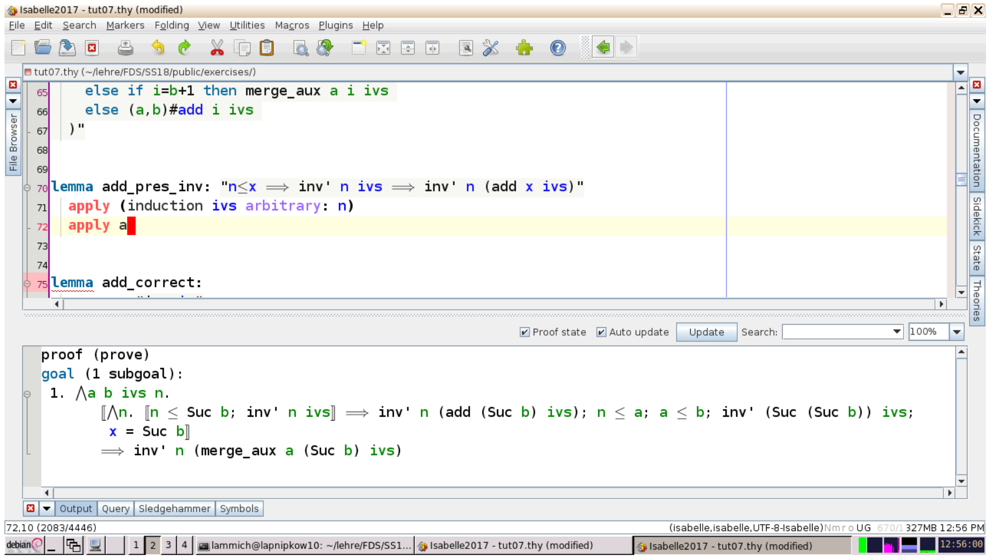

Isabelle2017 - tut07.thy
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
else if i≤b then (a,b)#ivs
else if i=b+1 then merge_aux a i ivs
else (a,b)#add i ivs
)"
lemma add_pres_inv: "n≤x ⟹ inv' n is ⟹ inv' n (add x is)"
unfolding inv_def
lemma add_correct:
assumes "inv is"
shows "inv (add x is)" "set_of (add x is) = {x} ∪ set_of is"
proof (prove)
goal (1 subgoal):
1. ⟦n ≤ x; inv' n is⟧ ⟹ inv' n (add x is)

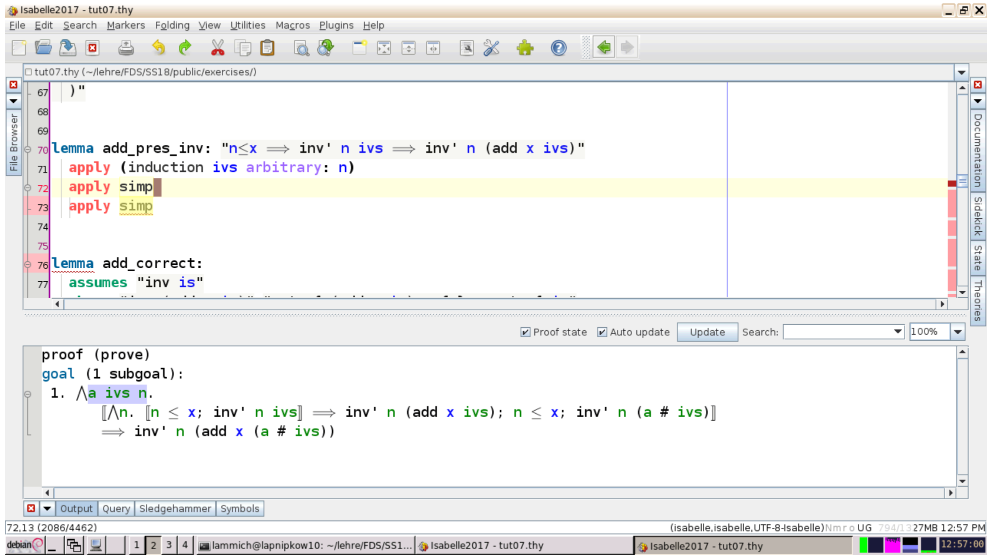

Isabelle2017 - tut07.thy
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
else if i≤b then (a,b)#ivs
else if i=b+1 then merge_aux a i ivs
else (a,b)#add i ivs
)"
lemma add_pres_inv: "n≤x ⟹ inv' n ivs ⟹ inv' n (add x ivs)"
unfolding inv_def
lemma add_correct:
assumes "inv is"
shows "inv (add x is)" "set_of (add x is) = {x} ∪ set_of is"
proof (prove)
goal (1 subgoal):
1. ⟦n ≤ x; inv' n ivs⟧ ⟹ inv' n (add x ivs)

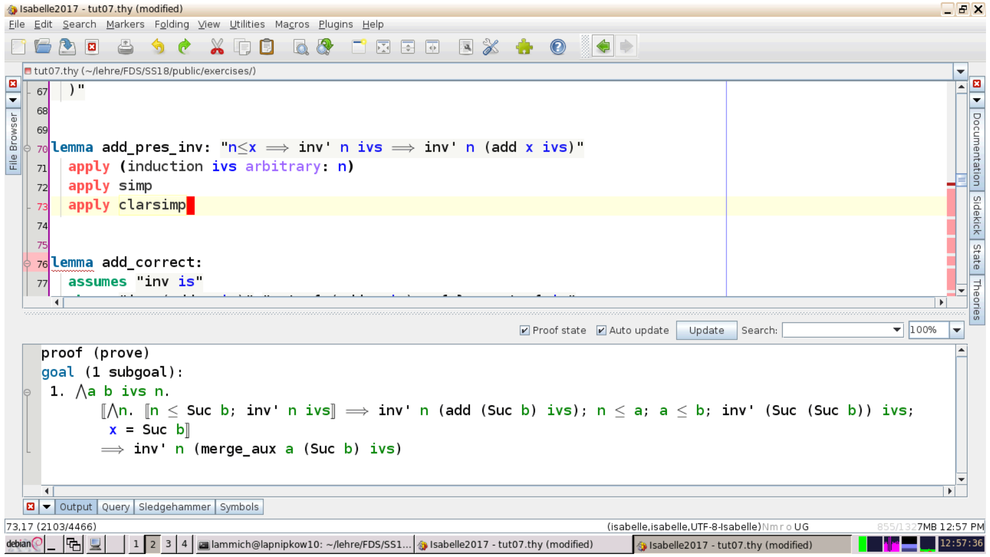
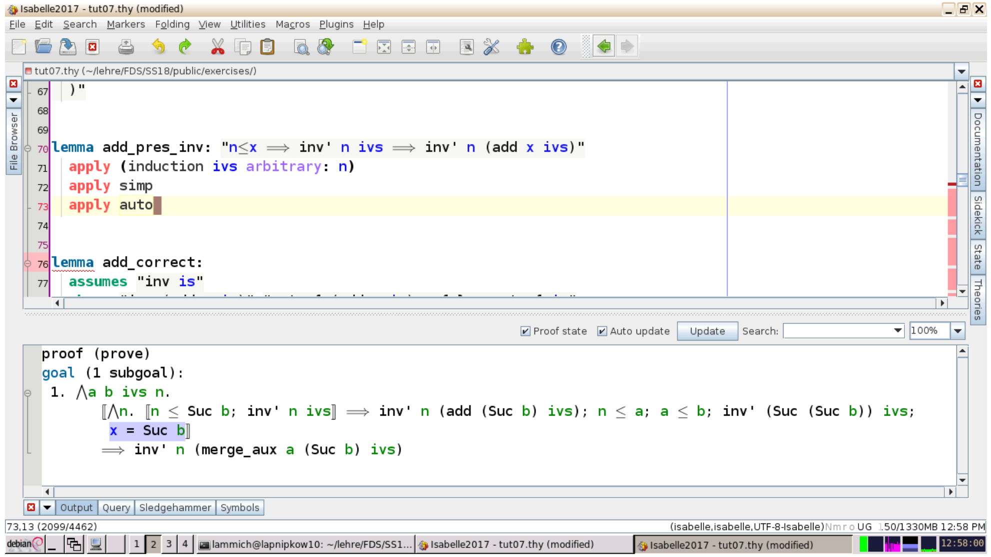
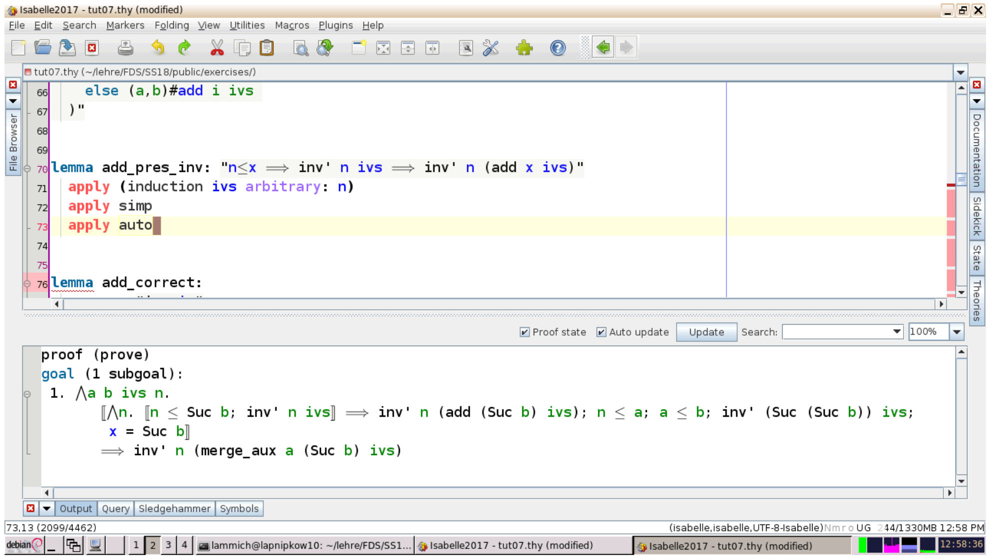
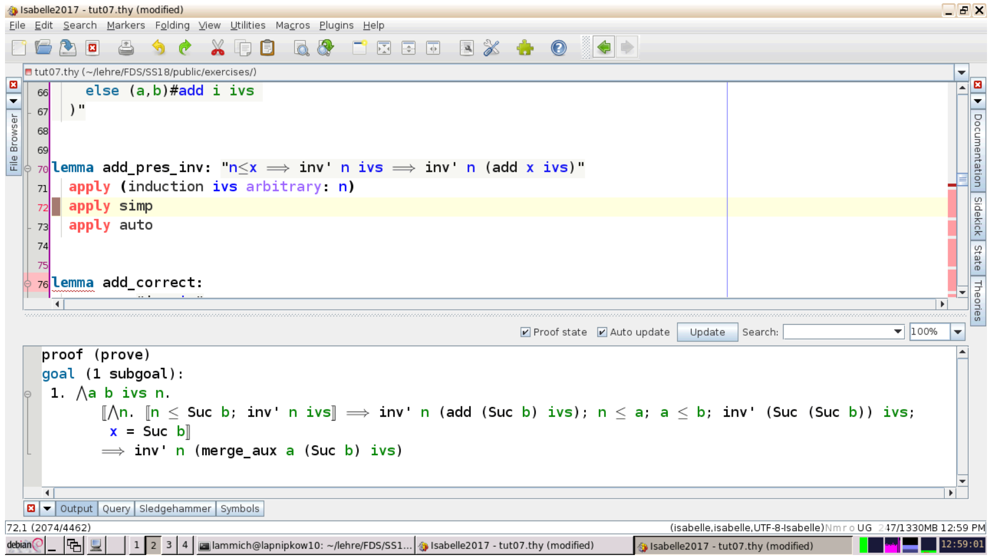

Isabelle2017 - tut07.thy
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
else if i≤b then (a,b)#ivs
else if i=b+1 then merge_aux a i ivs
else (a,b)#add i ivs
)"
lemma add_pres_inv: "n≤x ⟹ inv' n ivs ⟹ inv' n (add x ivs)"
apply (induction ivs)
lemma add_correct:
assumes "inv is"
shows "inv (add x is)" "set_of (add x is) = {x} ∪ set_of is"
proof (prove)
goal (2 subgoals):
1. ⟦n ≤ x; inv' n []⟧ ⟹ inv' n (add x [])
2. ⋀a ivs.
⟦⟦n ≤ x; inv' n ivs⟧ ⟹ inv' n (add x ivs); n ≤ x; inv' n (a # ivs)⟧ ⟹ inv' n (add x (a # ivs))

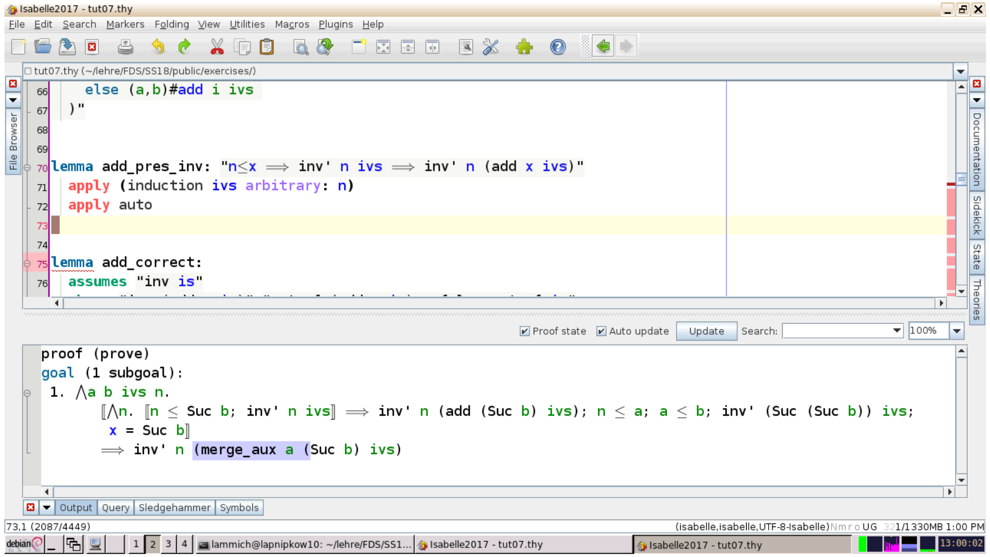

Isabelle2017 - tut07.thy
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
else if i≤b then (a,b)#ivs
else if i=b+1 then merge_aux a i ivs
else (a,b)#add i ivs
)"
lemma add_pres_inv: "n≤x ⟹ inv' n ivs ⟹ inv' n (add x ivs)"
apply (induction ivs)
lemma add_correct:
assumes "inv is"
shows "inv (add x is)" "set_of (add x is) = {x} ∪ set_of is"
proof (prove)
goal (2 subgoals):
1. ⟦n ≤ x; inv' n []⟧ ⟹ inv' n (add x [])
2. ⋀a ivs.
⟦⟦n ≤ x; inv' n ivs⟧ ⟹ inv' n (add x ivs); n ≤ x; inv' n (a # ivs)⟧ ⟹ inv' n (add x (a # ivs))

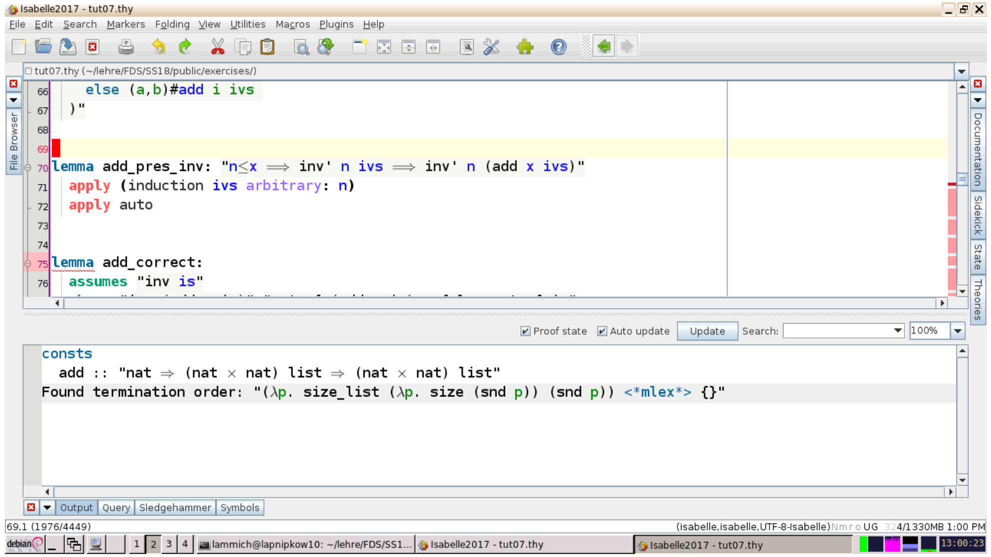

Isabelle2017 - tut07.thy
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
else if i≤b then (a,b)#ivs
else if i=b+1 then merge_aux a i ivs
else (a,b)#add i ivs
)"
lemma add_pres_inv: "n≤x ⟹ inv' n ivs ⟹ inv' n (add x ivs)"
apply (induction ivs arbitrary: n)
lemma add_correct:
proof (prove)
goal (2 subgoals):
1. ⋀n. ⟦n ≤ x; inv' n []⟧ ⟹ inv' n (add x [])
2. ⋀a ivs n.
⟦⋀n. ⟦n ≤ x; inv' n ivs⟧ ⟹ inv' n (add x ivs); n ≤ x; inv' n (a # ivs)⟧
⟹ inv' n (add x (a # ivs))

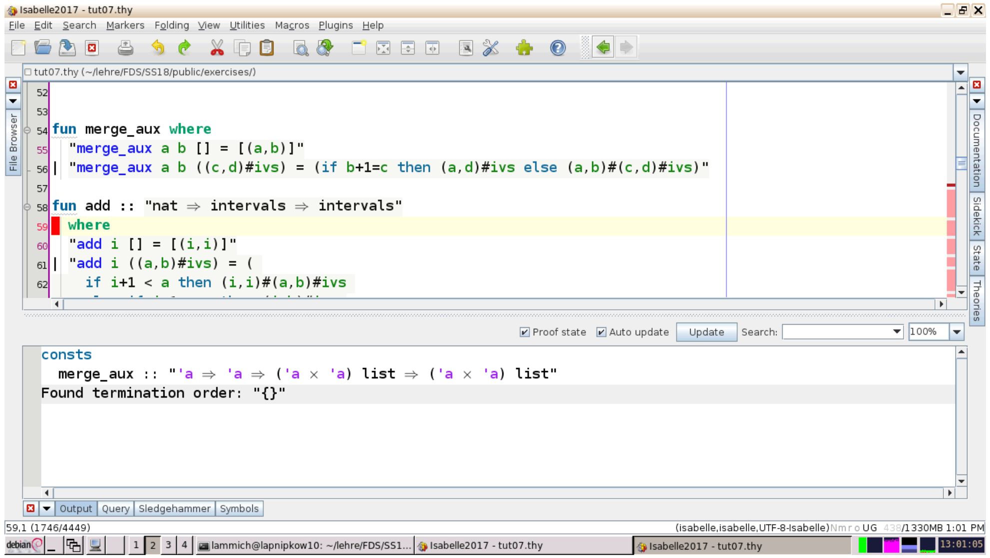

Isabelle2017 - tut07.thy (modified)
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
else if i=b+1 then merge_aux a i ivs
else (a,b)#add i ivs
)"
lemma add_pres_inv: "n≤x ⟹ inv' n ivs ⟹ inv' n (add x ivs)"
apply (induction ivs arbitrary: n)
apply a
lemma add_correct:
proof (prove)
goal (1 subgoal):
1. ⋀a b ivs n.
⟦⋀n. ⟦n ≤ Suc b; inv' n ivs⟧ ⟹ inv' n (add (Suc b) ivs); n ≤ a; a ≤ b; inv' (Suc (Suc b)) ivs;
x = Suc b⟧
⟹ inv' n (merge_aux a (Suc b) ivs)

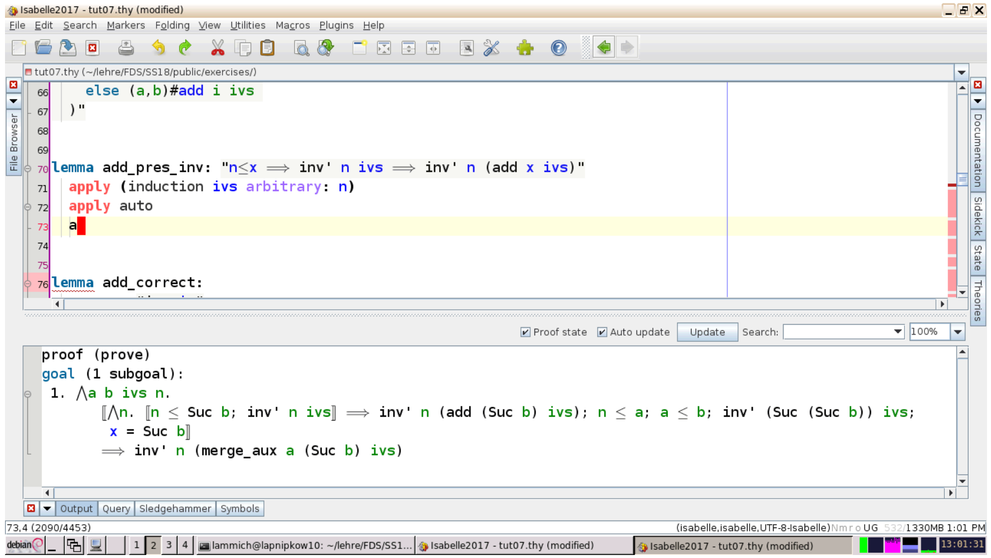

Isabelle2017 - tut07.thy
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
)"
lemma add_pres_inv: "n≤x ⟹ inv' n ivs ⟹ inv' n (add x ivs)"
apply (induction ivs arbitrary: n)
apply simp
apply simp
lemma add_correct:
assumes "inv is"
proof (prove)
goal (1 subgoal):
1. ⋀a ivs n.
⟦⋀n. ⟦n ≤ x; inv' n ivs⟧ ⟹ inv' n (add x ivs); n ≤ x; inv' n (a # ivs)⟧
⟹ inv' n (add x (a # ivs))

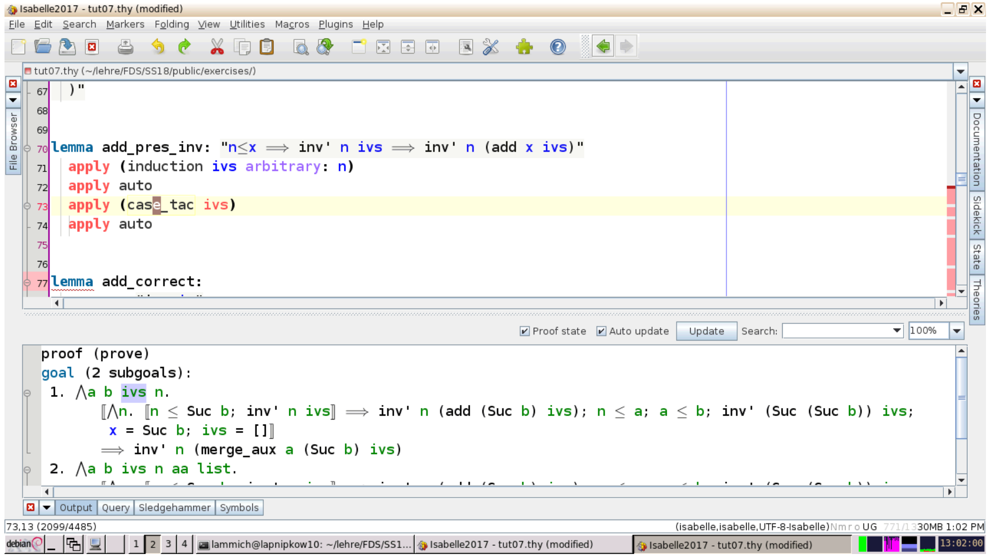

Isabelle2017 - tut07.thy (modified)
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
)"
lemma add_pres_inv: "n≤x ⟹ inv' n ivs ⟹ inv' n (add x ivs)"
apply (induction ivs arbitrary: n)
apply simp
apply clarsimp
lemma add_correct:
assumes "inv is"
proof (prove)
goal (1 subgoal):
1. ⋀a b ivs n.
⟦⋀n. ⟦n ≤ Suc b; inv' n ivs⟧ ⟹ inv' n (add (Suc b) ivs); n ≤ a; a ≤ b; inv' (Suc (Suc b)) ivs;
x = Suc b⟧
⟹ inv' n (merge_aux a (Suc b) ivs)

Isabelle2017 - tut07.thy (modified)
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
)"
lemma add_pres_inv: "n≤x ⟹ inv' n ivs ⟹ inv' n (add x ivs)"
apply (induction ivs arbitrary: n)
apply simp
apply auto
lemma add_correct:
assumes "inv is"
proof (prove)
goal (1 subgoal):
1. ⋀a b ivs n.
⟦⋀n. ⟦n ≤ Suc b; inv' n ivs⟧ ⟹ inv' n (add (Suc b) ivs); n ≤ a; a ≤ b; inv' (Suc (Suc b)) ivs;
x = Suc b⟧
⟹ inv' n (merge_aux a (Suc b) ivs)

Isabelle2017 - tut07.thy (modified)
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
else (a,b)#add i ivs
)"
lemma add_pres_inv: "n≤x ⟹ inv' n ivs ⟹ inv' n (add x ivs)"
apply (induction ivs arbitrary: n)
apply simp
apply auto
lemma add_correct:
proof (prove)
goal (1 subgoal):
1. ⋀a b ivs n.
⟦⋀n. ⟦n ≤ Suc b; inv' n ivs⟧ ⟹ inv' n (add (Suc b) ivs); n ≤ a; a ≤ b; inv' (Suc (Suc b)) ivs;
x = Suc b⟧
⟹ inv' n (merge_aux a (Suc b) ivs)

Isabelle2017 - tut07.thy (modified)
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
else (a,b)#add i ivs
)"
lemma add_pres_inv: "n≤x ⟹ inv' n ivs ⟹ inv' n (add x ivs)"
apply (induction ivs arbitrary: n)
apply simp
apply auto
lemma add_correct:
proof (prove)
goal (1 subgoal):
1. ⋀a b ivs n.
⟦⋀n. ⟦n ≤ Suc b; inv' n ivs⟧ ⟹ inv' n (add (Suc b) ivs); n ≤ a; a ≤ b; inv' (Suc (Suc b)) ivs;
x = Suc b⟧
⟹ inv' n (merge_aux a (Suc b) ivs)

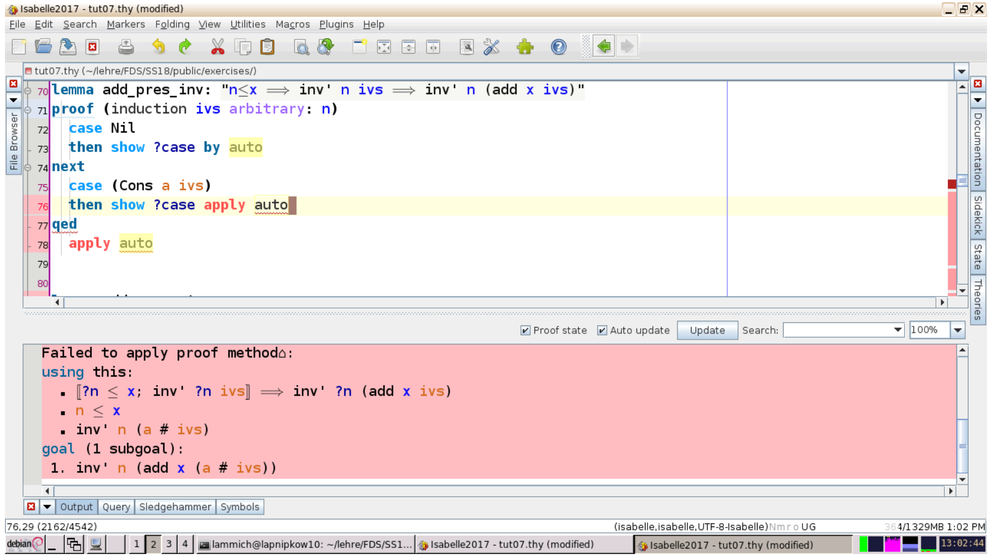

Isabelle2017 - tut07.thy
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
else (a,b)#add i ivs
)"
lemma add_pres_inv: "n≤x ⟹ inv' n ivs ⟹ inv' n (add x ivs)"
apply (induction ivs arbitrary: n)
apply auto
lemma add_correct:
assumes "inv is"
proof (prove)
goal (1 subgoal):
1. ⋀a b ivs n.
⟦⋀n. ⟦n ≤ Suc b; inv' n ivs⟧ ⟹ inv' n (add (Suc b) ivs); n ≤ a; a ≤ b; inv' (Suc (Suc b)) ivs;
x = Suc b⟧
⟹ inv' n (merge_aux a (Suc b) ivs)

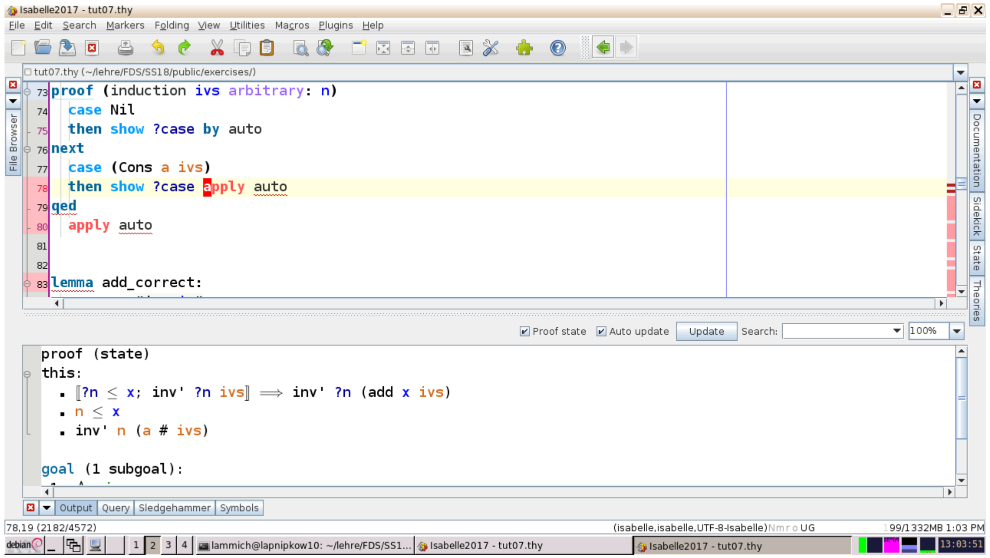

Isabelle2017 - tut07.thy
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
else (a,b)#add i ivs
)"
lemma add_pres_inv: "n≤x ⟹ inv' n ivs ⟹ inv' n (add x ivs)"
apply (induction ivs arbitrary: n)
apply auto
lemma add_correct:
assumes "inv is"
consts
add :: "nat ⇒ (nat × nat) list ⇒ (nat × nat) list"
Found termination order: "(λp. size_list (λp. size (snd p)) (snd p)) <*mlex*> {}"

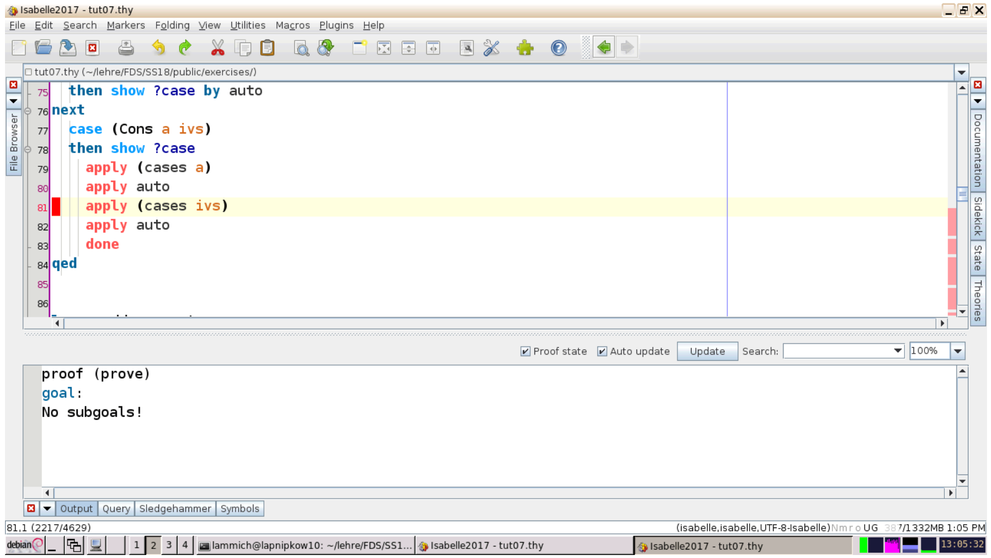

Isabelle2017 - tut07.thy
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
fun merge_aux where
"merge_aux a b [] = [(a,b)]"
| "merge_aux a b ((c,d)#ivs) = (if b+1=c then (a,d)#ivs else (a,b)#(c,d)#ivs)"
fun add :: "nat ⇒ intervals ⇒ intervals"
where
"add i [] = [(i,i)]"
| "add i ((a,b)#ivs) = (
if i+1 < a then (i,i)#(a,b)#ivs
consts
merge_aux :: "'a ⇒ 'a ⇒ ('a × 'a) list ⇒ ('a × 'a) list"
Found termination order: "{}"

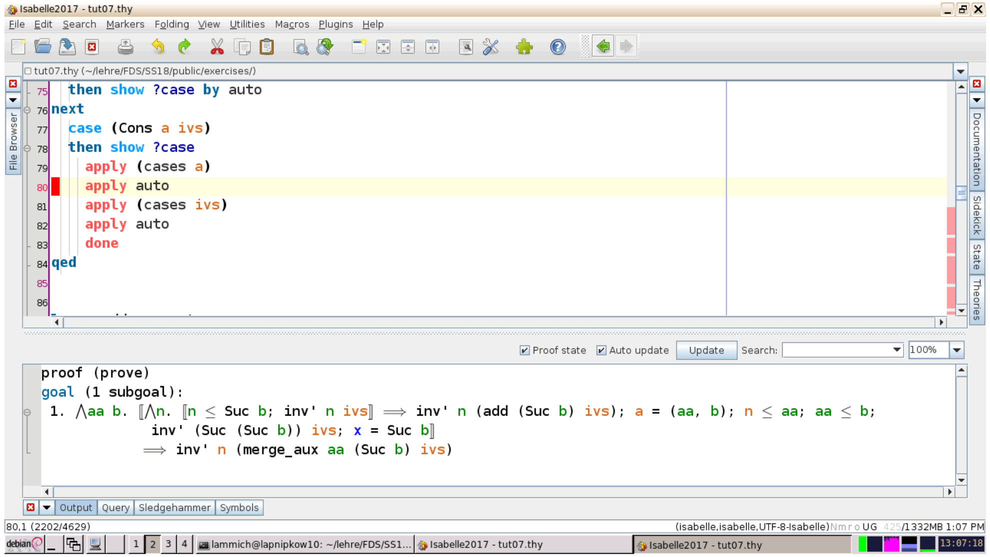

Isabelle2017 - tut07.thy (modified)
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
else (a,b)#add i ivs
)"
lemma add_pres_inv: "n≤x ⟹ inv' n ivs ⟹ inv' n (add x ivs)"
apply (induction ivs arbitrary: n)
apply auto
a
lemma add_correct:
proof (prove)
goal (1 subgoal):
1. ⋀a b ivs n.
⟦⋀n. ⟦n ≤ Suc b; inv' n ivs⟧ ⟹ inv' n (add (Suc b) ivs); n ≤ a; a ≤ b; inv' (Suc (Suc b)) ivs;
x = Suc b⟧
⟹ inv' n (merge_aux a (Suc b) ivs)

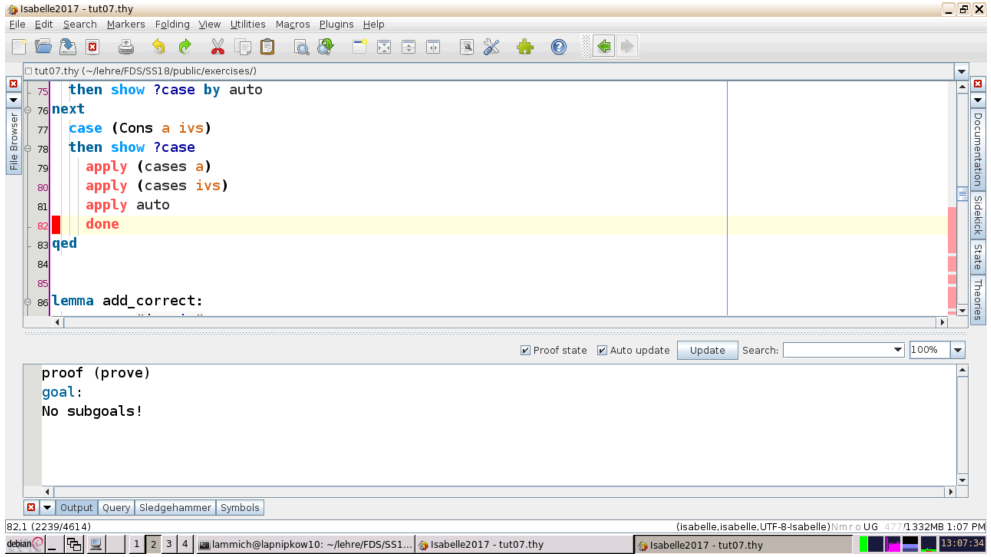

Isabelle2017 - tut07.thy (modified)
lemma add_pres_inv: "n≤x ⟹ inv' n ivs ⟹ inv' n (add x ivs)"
apply (induction ivs arbitrary: n)
apply auto
apply (case_tac ivs)
apply auto
lemma add_correct:
proof (prove)
goal (2 subgoals):
1. ⋀a b ivs n.
⟦⋀n. ⟦n ≤ Suc b; inv' n ivs⟧ ⟹ inv' n (add (Suc b) ivs); n ≤ a; a ≤ b; inv' (Suc (Suc b)) ivs;
x = Suc b; ivs = []⟧
⟹ inv' n (merge_aux a (Suc b) ivs)
2. ⋀a b ivs n aa list.

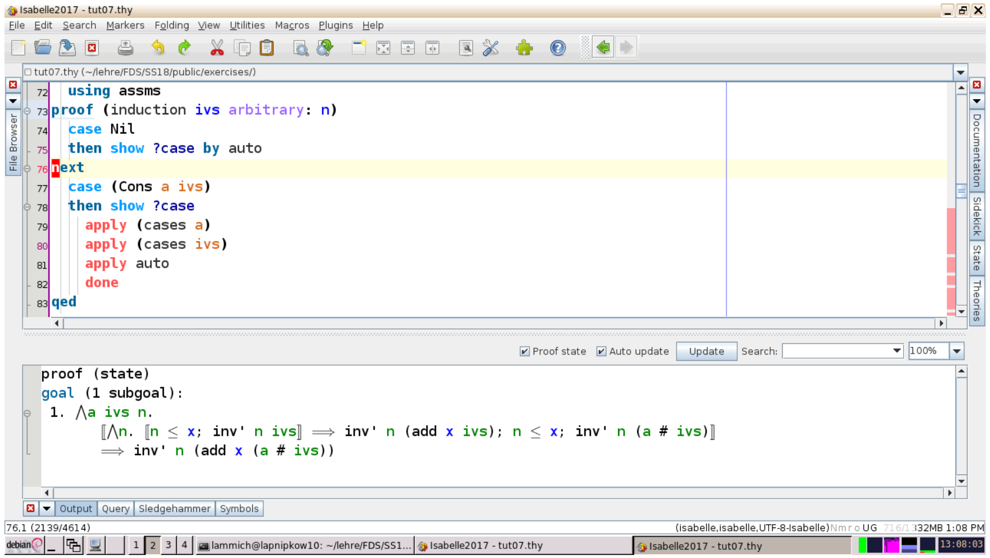

Isabelle2017 - tut07.thy (modified)
lemma add_pres_inv: "n≤x ⟹ inv' n ivs ⟹ inv' n (add x ivs)"
proof (induction ivs arbitrary: n)
case Nil
then show ?case by auto
next
case (Cons a ivs)
then show ?case apply auto
qed
apply auto
Failed to apply proof method:
using this:
⟦?n ≤ x; inv' ?n ivs⟧ ⟹ inv' ?n (add x ivs)
n ≤ x
inv' n (a # ivs)
goal (1 subgoal):
1. inv' n (add x (a # ivs))

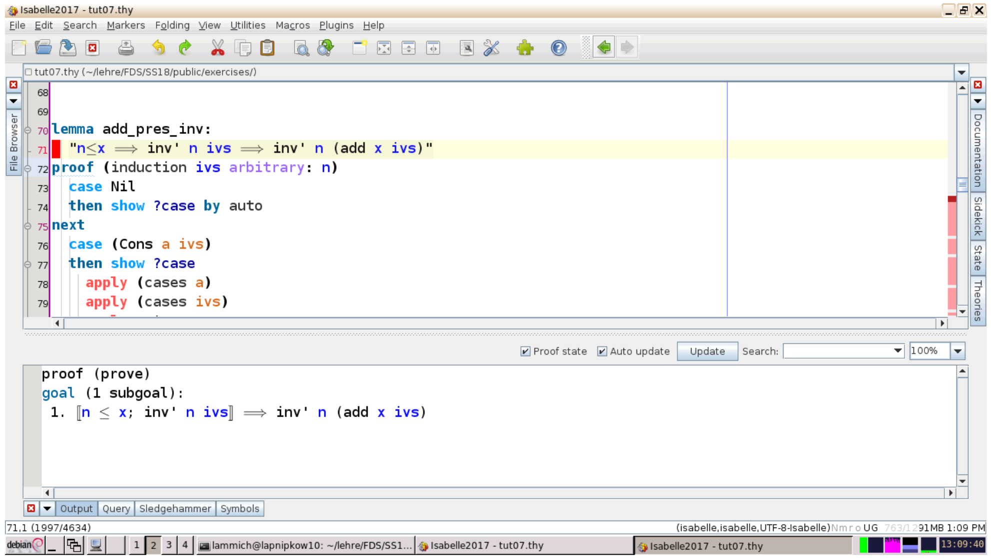

Isabelle2017 - tut07.thy
proof (induction ivs arbitrary: n)
case Nil
then show ?case by auto
next
case (Cons a ivs)
then show ?case apply auto
qed
apply auto
lemma add_correct:
proof (state)
this:
⟦?n ≤ x; inv' ?n ivs⟧ ⟹ inv' ?n (add x ivs)
n ≤ x
inv' n (a # ivs)
goal (1 subgoal):

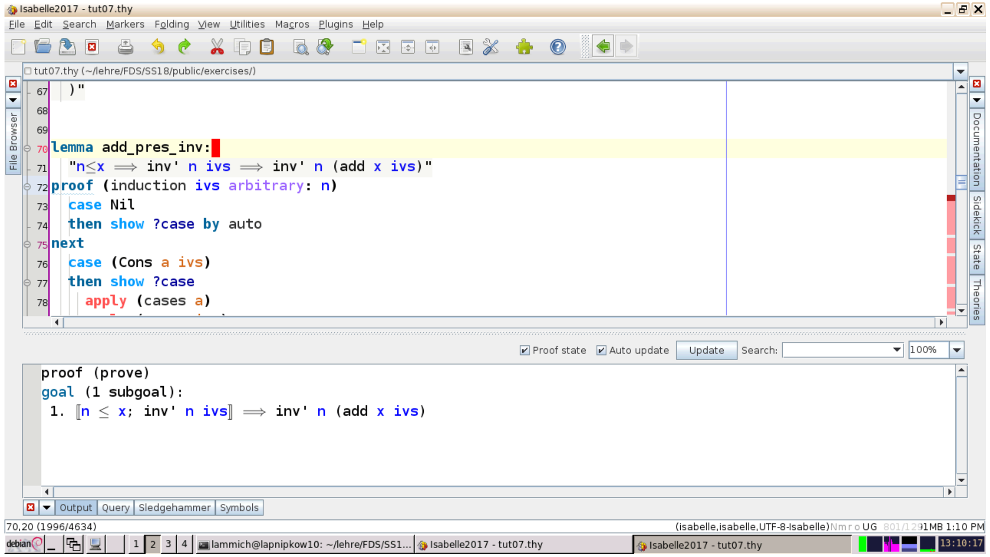

Isabelle2017 - tut07.thy
then show ?case by auto
next
case (Cons a ivs)
then show ?case
apply (cases a)
apply auto
apply (cases ivs)
apply auto
done
qed
proof (prove)
goal:
No subgoals!

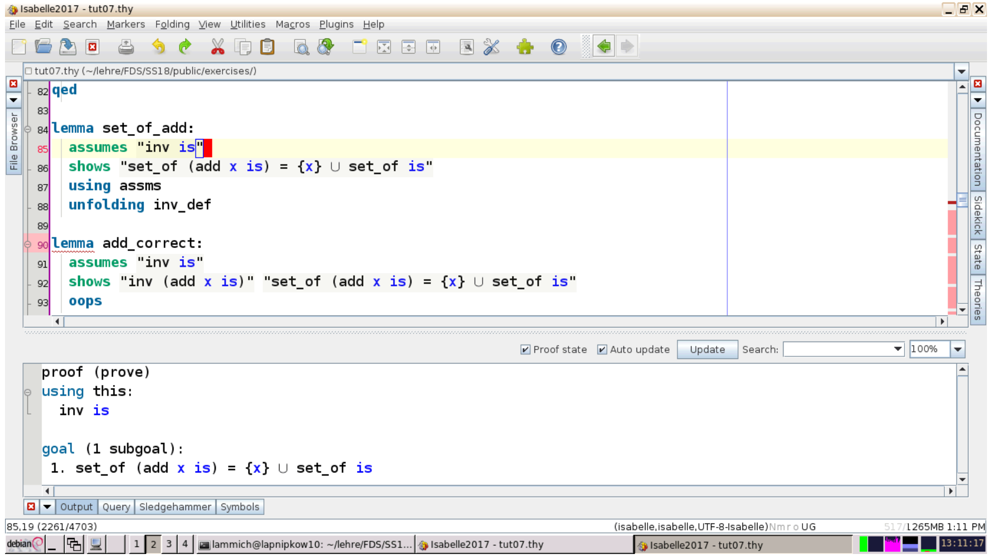

Isabelle2017 - tut07.thy
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
then show ?case by auto
next
case (Cons a ivs)
then show ?case
apply (cases a)
apply auto
apply (cases ivs)
apply auto
done
qed
proof (prove)
goal (1 subgoal):
1. ⋀aa b. ⟦⋀n. ⟦n ≤ Suc b; inv' n ivs⟧ ⟹ inv' n (add (Suc b) ivs); a = (aa, b); n ≤ aa; aa ≤ b;
inv' (Suc (Suc b)) ivs; x = Suc b⟧
⟹ inv' n (merge_aux aa (Suc b) ivs)

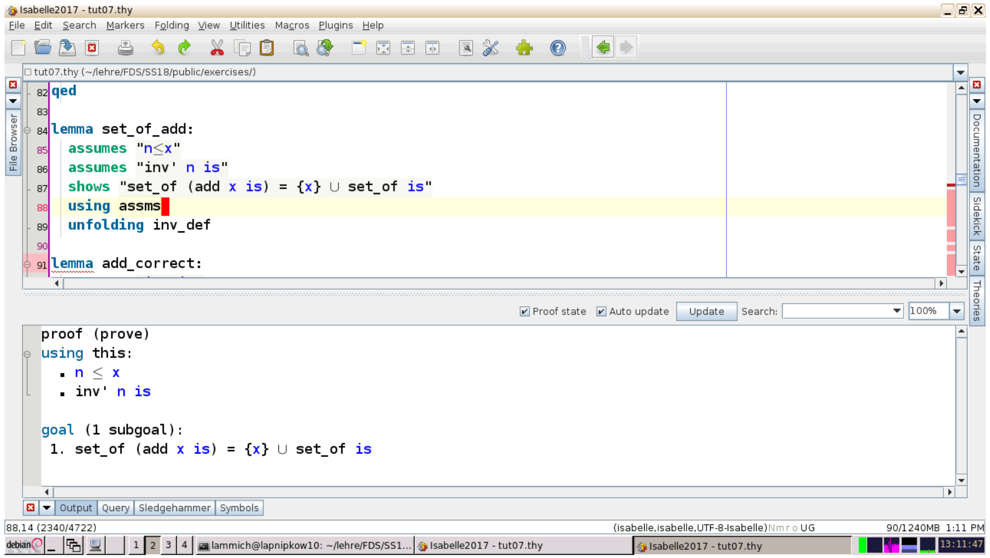

Isabelle2017 - tut07.thy
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
then show ?case by auto
next
case (Cons a ivs)
then show ?case
apply (cases a)
apply (cases ivs)
apply auto
done
qed
lemma add_correct:
proof (prove)
goal:
No subgoals!

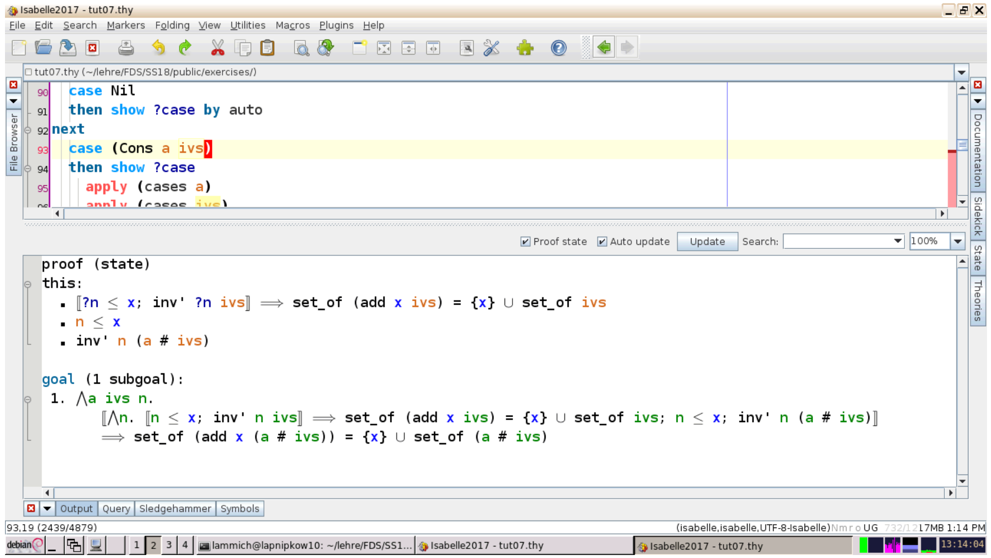

Isabelle2017 - tut07.thy
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
using assms
proof (induction ivs arbitrary: n)
case Nil
then show ?case by auto
next
case (Cons a ivs)
then show ?case
apply (cases a)
apply (cases ivs)
apply auto
done
qed
proof (state)
goal (1 subgoal):
1. ⋀a ivs n.
⟦⋀n. ⟦n ≤ x; inv' n ivs⟧ ⟹ inv' n (add x ivs); n ≤ x; inv' n (a # ivs)⟧
⟹ inv' n (add x (a # ivs))

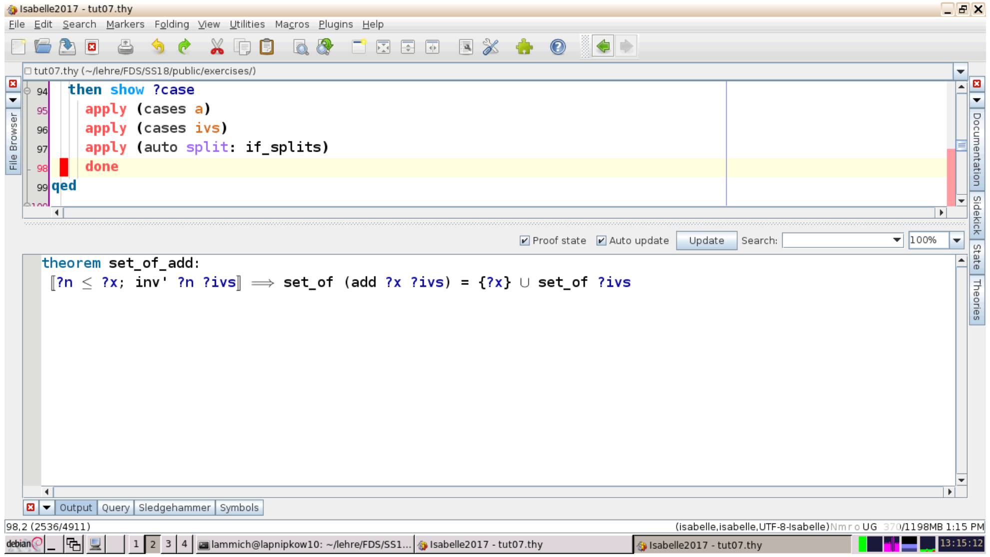

Isabelle2017 - tut07.thy
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
lemma add_pres_inv:
"n≤x ⟹ inv' n ivs ⟹ inv' n (add x ivs)"
proof (induction ivs arbitrary: n)
case Nil
then show ?case by auto
next
case (Cons a ivs)
then show ?case
apply (cases a)
apply (cases ivs)
proof (prove)
goal (1 subgoal):
1. ⟦n ≤ x; inv' n ivs⟧ ⟹ inv' n (add x ivs)

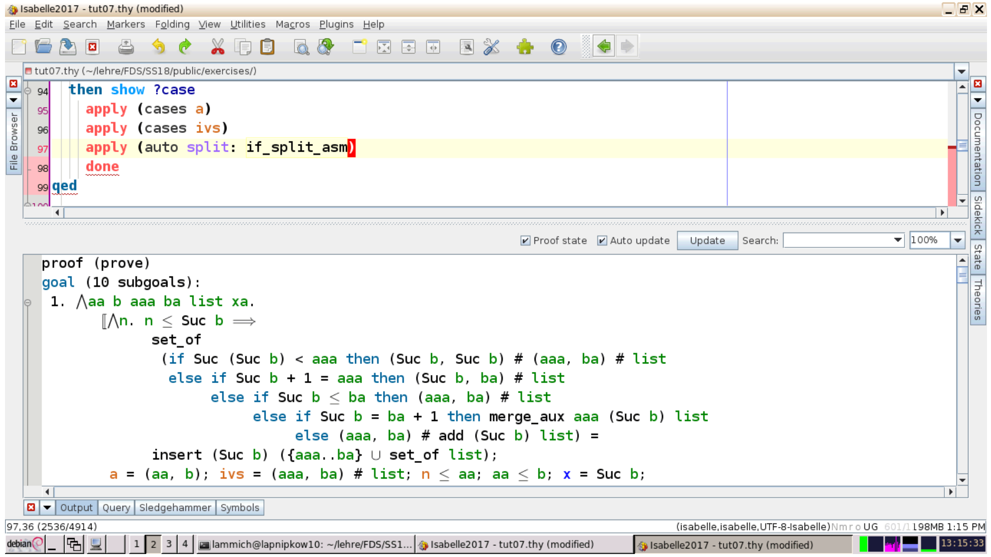

Isabelle2017 - tut07.thy
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
)"
lemma add_pres_inv:
"n≤x ⟹ inv' n ivs ⟹ inv' n (add x ivs)"
proof (induction ivs arbitrary: n)
case Nil
then show ?case by auto
next
case (Cons a ivs)
then show ?case
apply (cases a)
proof (prove)
goal (1 subgoal):
1. ⟦n ≤ x; inv' n ivs⟧ ⟹ inv' n (add x ivs)

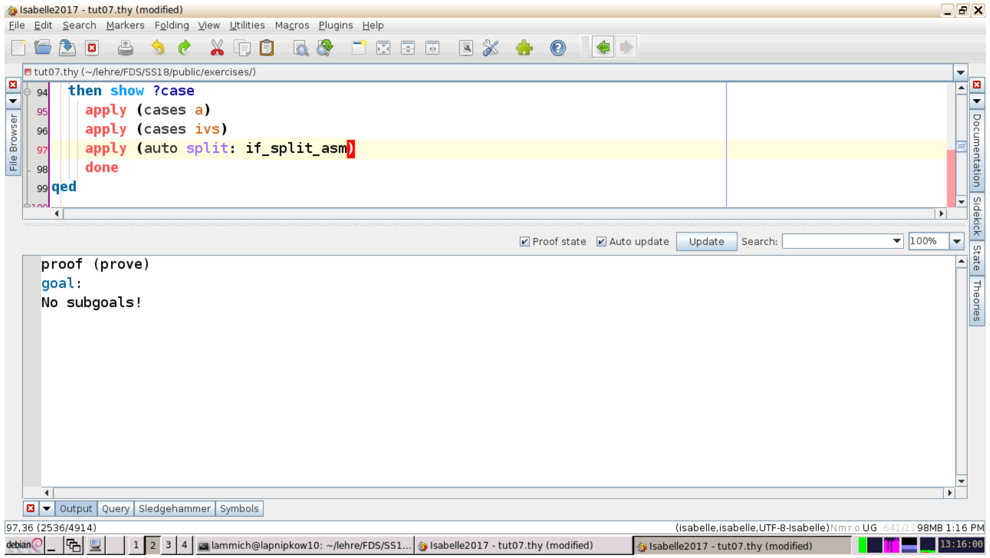

Isabelle2017 - tut07.thy
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
qed
lemma set_of_add:
assumes "inv is"
shows "set_of (add x is) = {x} ∪ set_of is"
using assms
unfolding inv_def
lemma add_correct:
assumes "inv is"
shows "inv (add x is)" "set_of (add x is) = {x} ∪ set_of is"
oops
proof (prove)
using this:
inv is
goal (1 subgoal):
1. set_of (add x is) = {x} ∪ set_of is

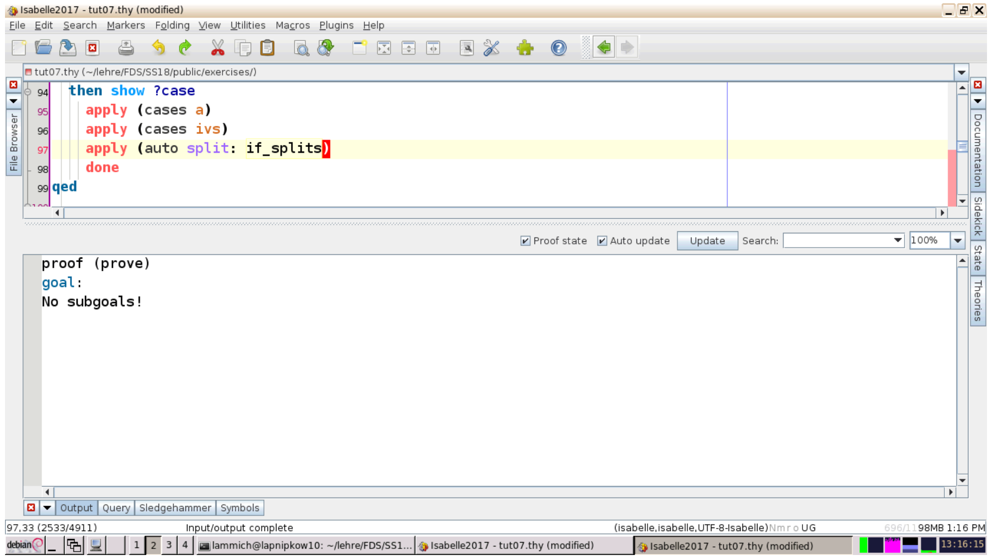

Isabelle2017 - tut07.thy
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
qed
lemma set_of_add:
assumes "n≤x"
assumes "inv' n is"
shows "set_of (add x is) = {x} ∪ set_of is"
using assms
unfolding inv_def
lemma add_correct:
proof (prove)
using this:
n ≤ x
inv' n is
goal (1 subgoal):
1. set_of (add x is) = {x} ∪ set_of is

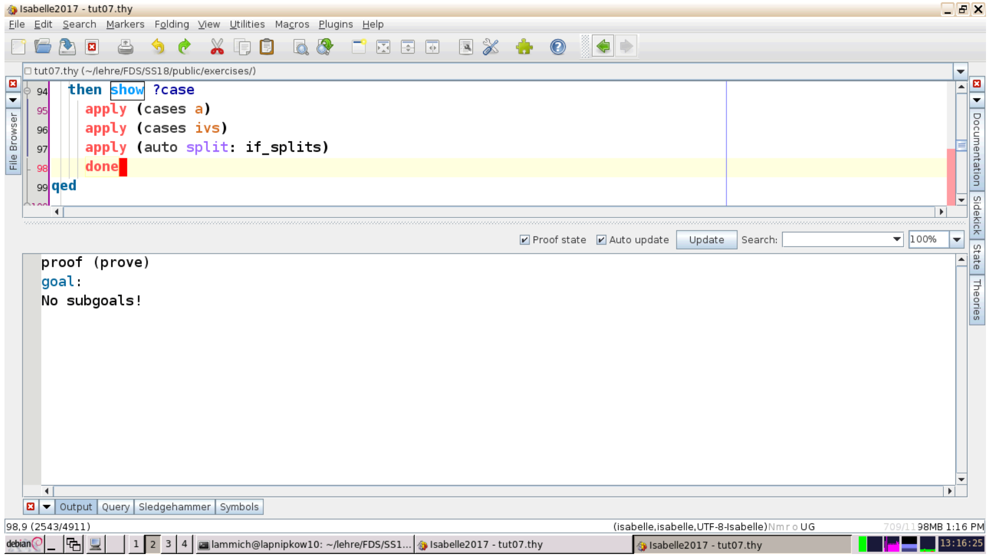

Isabelle2017 - tut07.thy
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
case Nil
then show ?case by auto
next
case (Cons a ivs)
then show ?case
apply (cases a)
proof (state)
this:
⟦?n ≤ x; inv' ?n ivs⟧ ⟹ set_of (add x ivs) = {x} ∪ set_of ivs
n ≤ x
inv' n (a # ivs)
goal (1 subgoal):
1. ⋀a ivs n.
⟦⋀n. ⟦n ≤ x; inv' n ivs⟧ ⟹ set_of (add x ivs) = {x} ∪ set_of ivs; n ≤ x; inv' n (a # ivs)⟧
⟹ set_of (add x (a # ivs)) = {x} ∪ set_of (a # ivs)

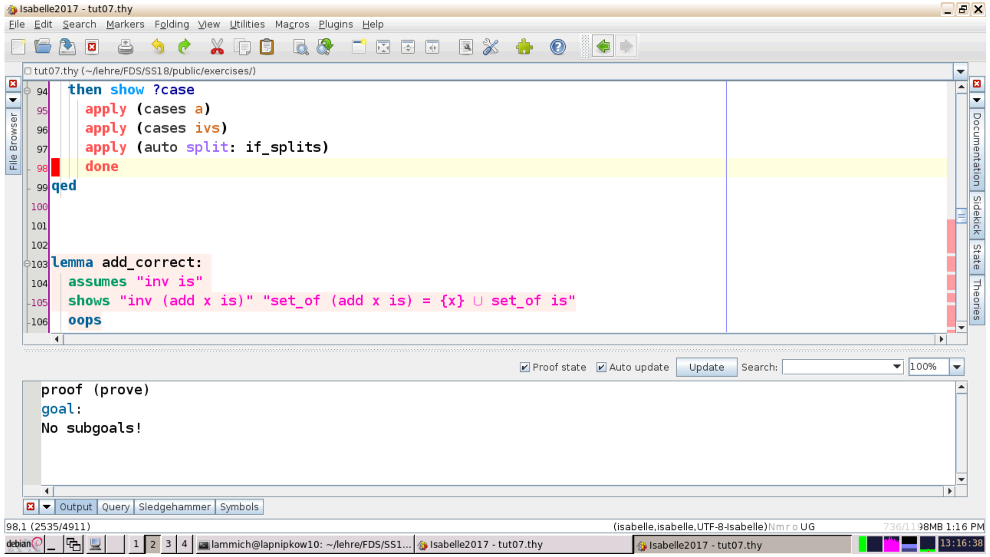

Isabelle2017 - tut07.thy
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
then show ?case
apply (cases a)
apply (cases ivs)
apply (auto split: if_splits)
done
qed
theorem set_of_add:
⟦?n ≤ ?x; inv' ?n ?ivs⟧ ⟹ set_of (add ?x ?ivs) = {?x} ∪ set_of ?ivs

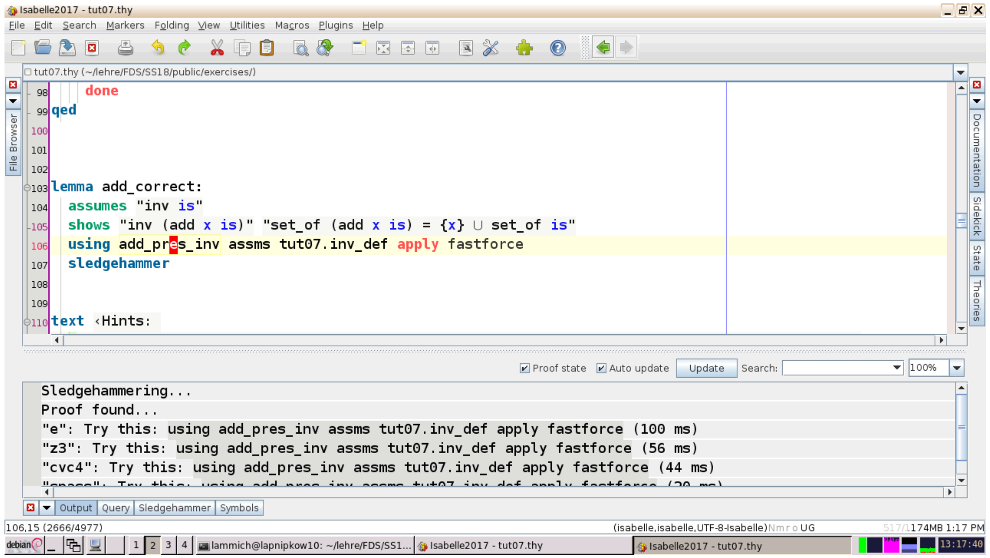

Isabelle2017 - tut07.thy (modified)
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
then show ?case
apply (cases a)
apply (cases ivs)
apply (auto split: if_split_asm)
done
qed
proof (prove)
goal (10 subgoals):
1. ⋀aa b aaa ba list xa.
⟦⋀n. n ≤ Suc b ⟹
set_of
(if Suc (Suc b) < aaa then (Suc b, Suc b) # (aaa, ba) # list
else if Suc b + 1 = aaa then (Suc b, ba) # list
else if Suc b ≤ ba then (aaa, ba) # list
else if Suc b = ba + 1 then merge_aux aaa (Suc b) list
else (aaa, ba) # add (Suc b) list) =
insert (Suc b) ({aaa..ba} ∪ set_of list);
a = (aa, b); ivs = (aaa, ba) # list; n ≤ aa; aa ≤ b; x = Suc b;

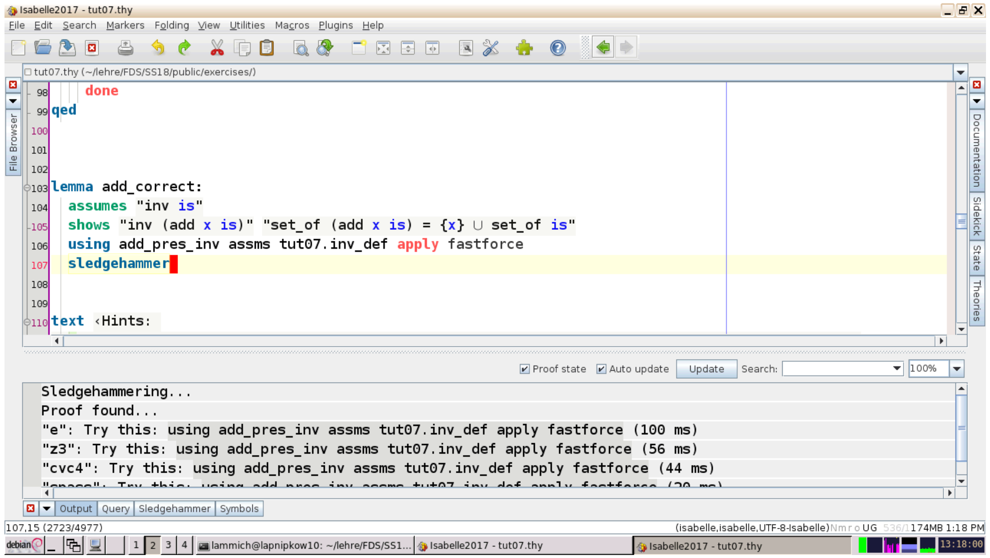

Isabelle2017 - tut07.thy (modified)
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
then show ?case
apply (cases a)
apply (cases ivs)
apply (auto split: if_split_asm)
done
qed
proof (prove)
goal:
No subgoals!

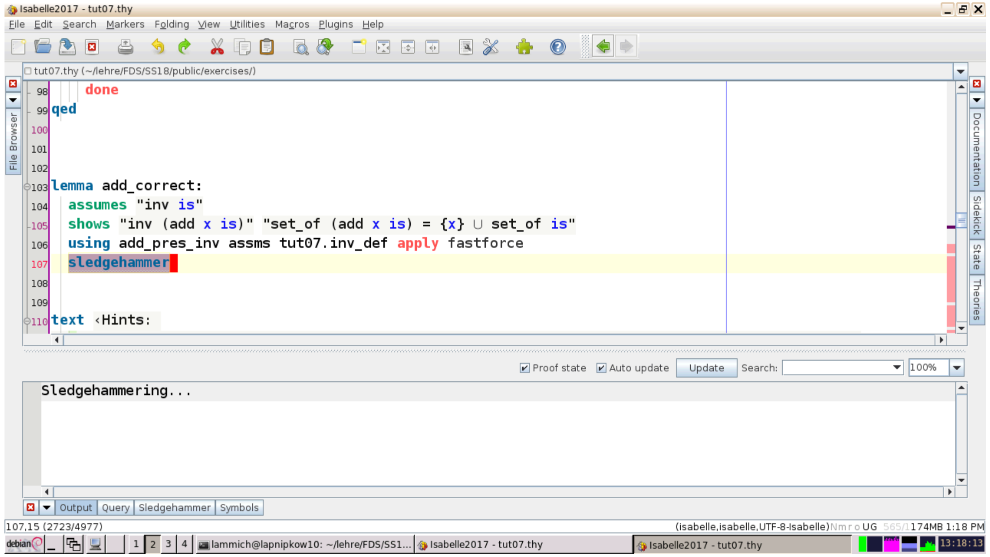

Isabelle2017 - tut07.thy (modified)
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
then show ?case
apply (cases a)
apply (cases ivs)
apply (auto split: if_splits)
done
qed
proof (prove)
goal:
No subgoals!

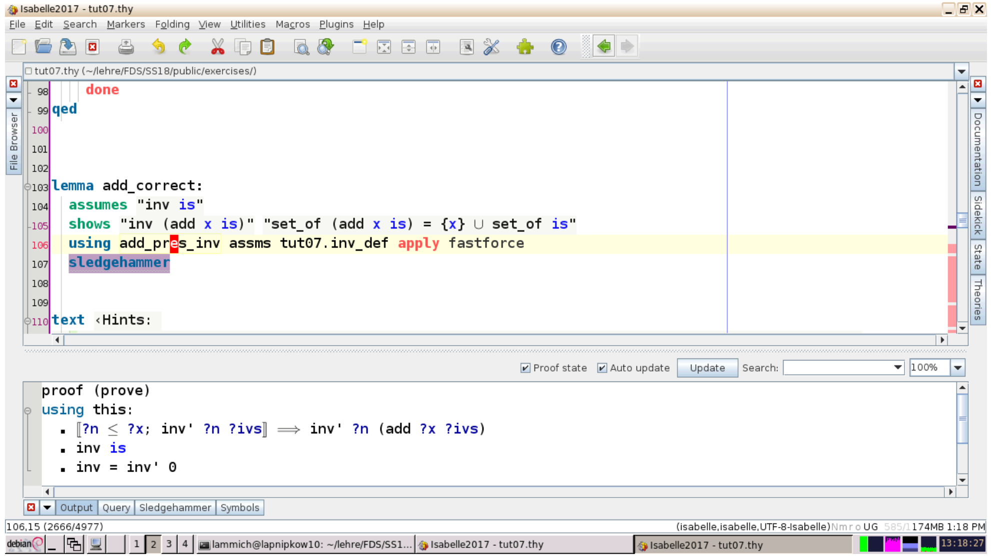

Isabelle2017 - tut07.thy
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
then show ?case
apply (cases a)
apply (cases ivs)
apply (auto split: if_splits)
done
qed
proof (prove)
goal:
No subgoals!
Output Query Sledgehammer Symbols

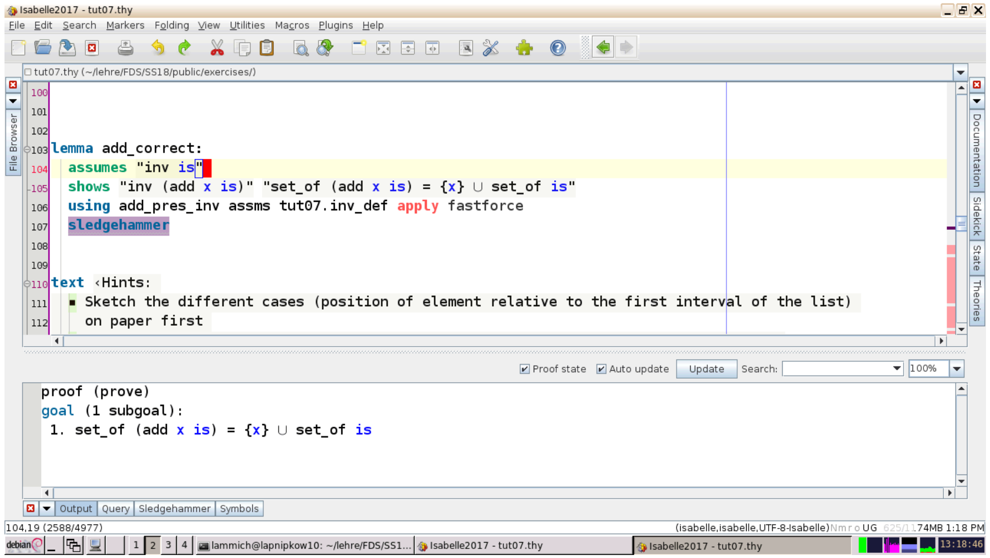

Isabelle2017 - tut07.thy
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
then show ?case
apply (cases a)
apply (cases ivs)
apply (auto split: if_splits)
done
qed
lemma add_correct:
assumes "inv is"
shows "inv (add x is)" "set_of (add x is) = {x} ∪ set_of is"
oops
proof (prove)
goal:
No subgoals!
Output Query Sledgehammer Symbols

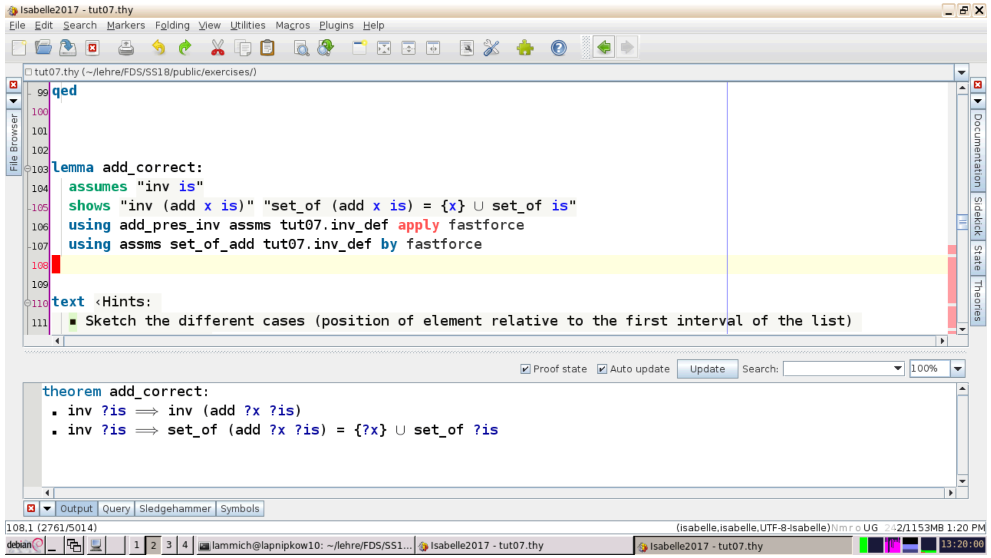

Isabelle2017 - tut07.thy
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
done
qed
lemma add_correct:
assumes "inv is"
shows "inv (add x is)" "set_of (add x is) = {x} ∪ set_of is"
using add_pres_inv assms tut07.inv_def apply fastforce
sledgehammer
text ‹Hints:
Sledgehammering...
Proof found...
"e": Try this: using add_pres_inv assms tut07.inv_def apply fastforce (100 ms)
"z3": Try this: using add_pres_inv assms tut07.inv_def apply fastforce (56 ms)
"cvc4": Try this: using add_pres_inv assms tut07.inv_def apply fastforce (44 ms)
Output Query Sledgehammer Symbols

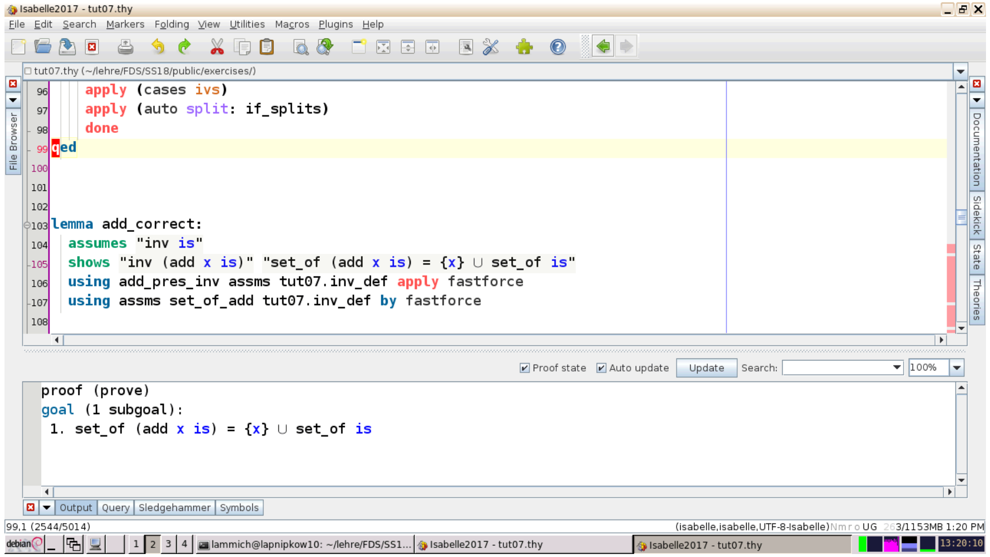
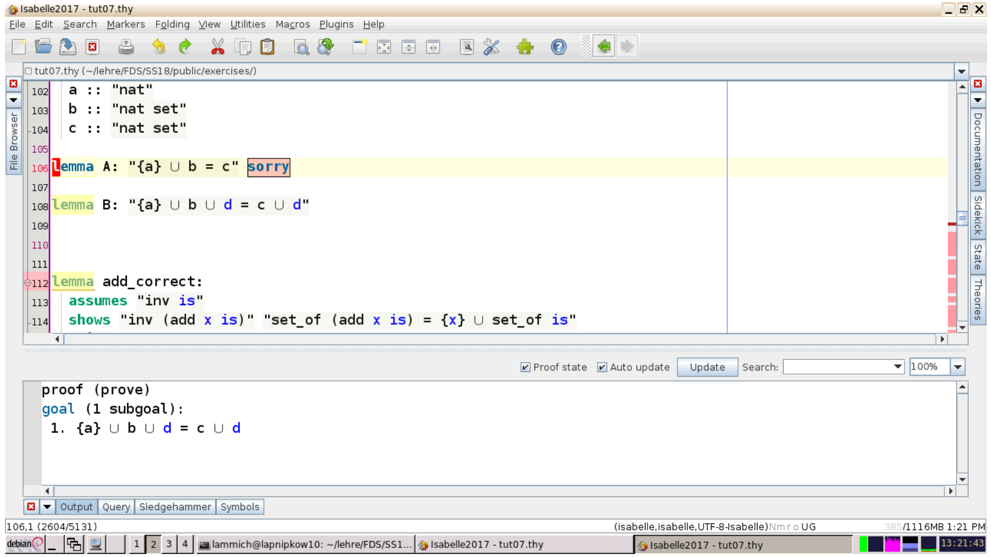
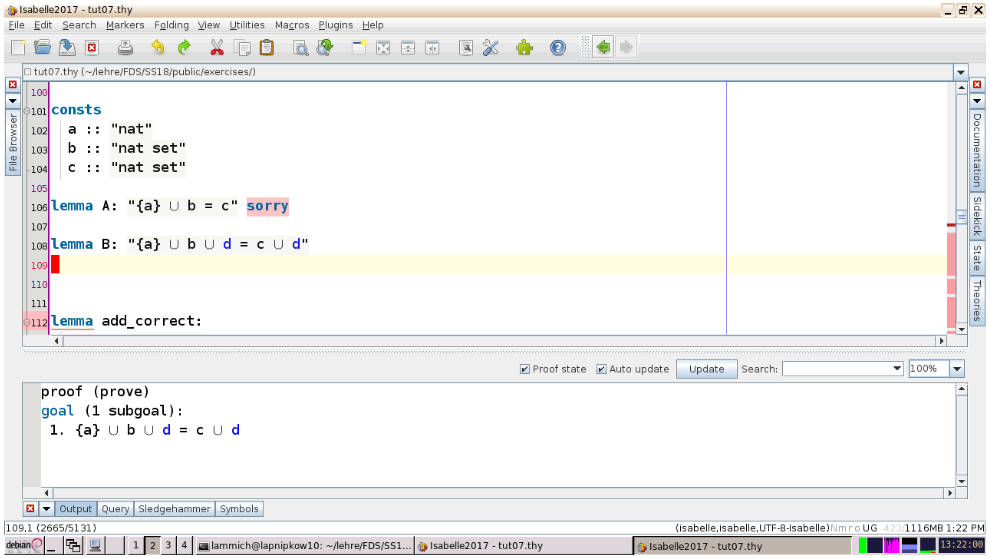
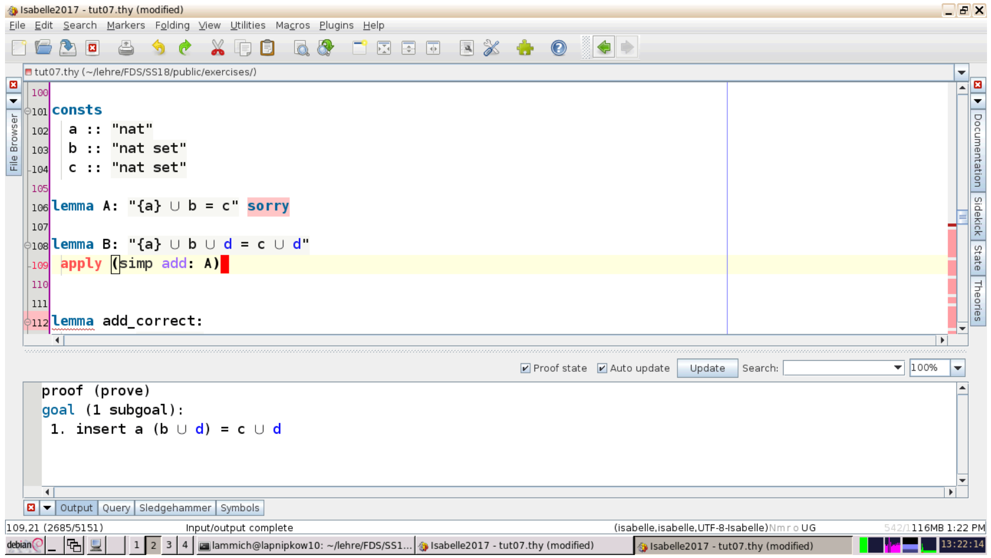

Isabelle2017 - tut07.thy
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
done
qed
lemma add_correct:
assumes "inv is"
shows "inv (add x is)" "set_of (add x is) = {x} ∪ set_of is"
using add_pres_inv assms tut07.inv_def apply fastforce
sledgehammer
text ‹Hints:
Sledgehammering...
Proof found...
"e": Try this: using add_pres_inv assms tut07.inv_def apply fastforce (100 ms)
"z3": Try this: using add_pres_inv assms tut07.inv_def apply fastforce (56 ms)
"cvc4": Try this: using add_pres_inv assms tut07.inv_def apply fastforce (44 ms)
Output Query Sledgehammer Symbols

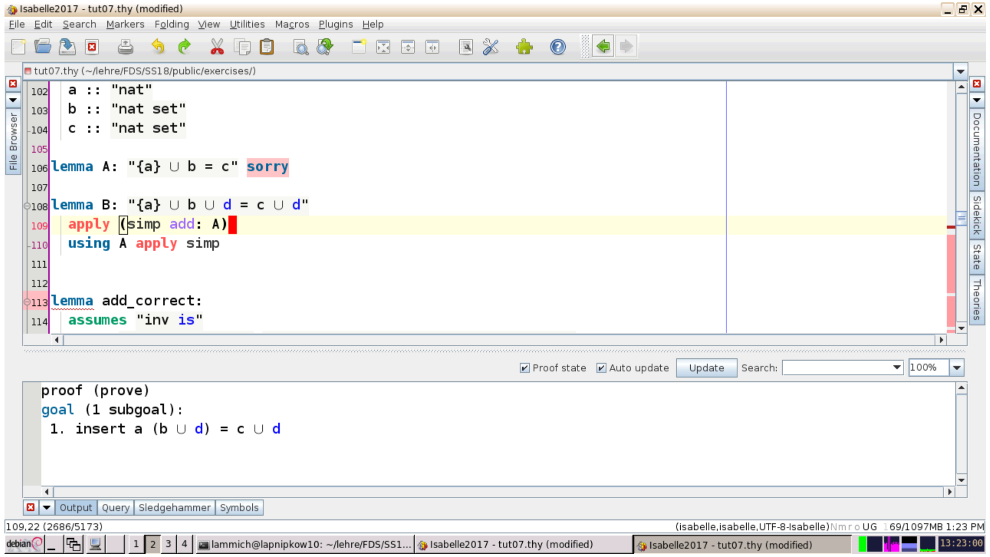

Isabelle2017 - tut07.thy
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
done
qed
lemma add_correct:
assumes "inv is"
shows "inv (add x is)" "set_of (add x is) = {x} ∪ set_of is"
using add_pres_inv assms tut07.inv_def apply fastforce
sledgehammer
text ‹Hints:
Sledgehammering...

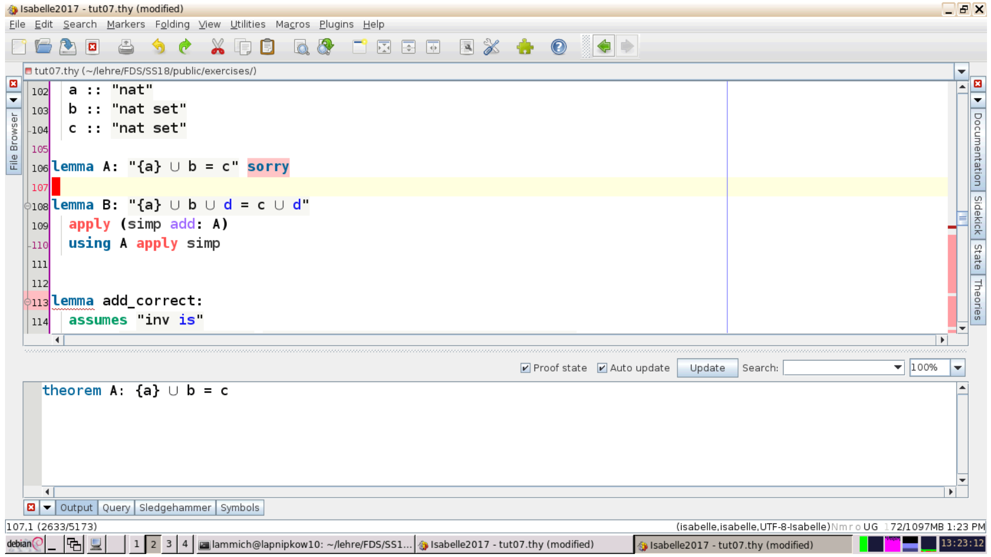

Isabelle2017 - tut07.thy
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
done
qed
lemma add_correct:
assumes "inv is"
shows "inv (add x is)" "set_of (add x is) = {x} ∪ set_of is"
using add_pres_inv assms tut07.inv_def apply fastforce
sledgehammer
text ‹Hints:
proof (prove)
using this:
⟦?n ≤ ?x; inv' ?n ?ivs⟧ ⟹ inv' ?n (add ?x ?ivs)
inv is
inv = inv' 0

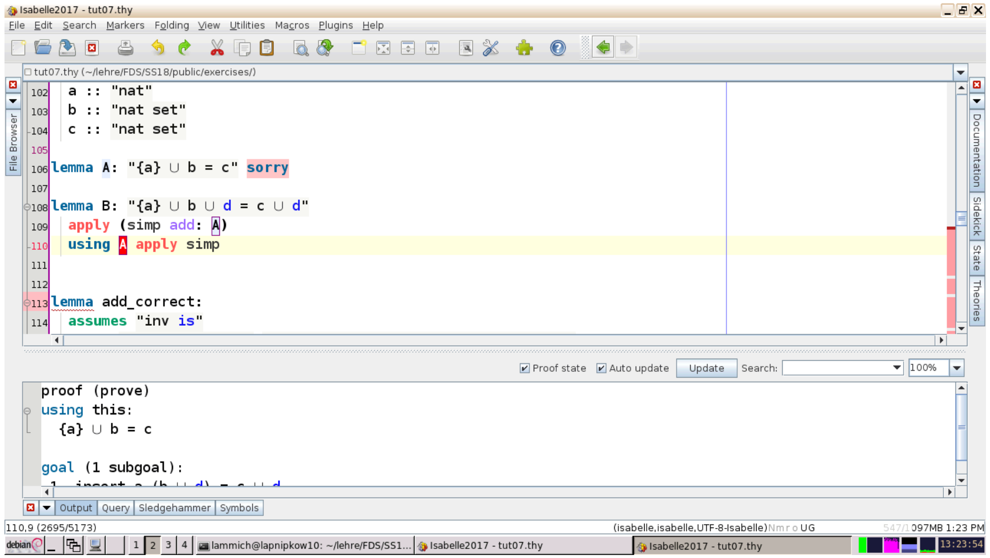

Isabelle2017 - tut07.thy
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
lemma add_correct:
assumes "inv is"
shows "inv (add x is)" "set_of (add x is) = {x} ∪ set_of is"
using add_pres_inv assms tut07.inv_def apply fastforce
sledgehammer
text ‹Hints:
Sketch the different cases (position of element relative to the first interval of the list)
on paper first
proof (prove)
goal (1 subgoal):
1. set_of (add x is) = {x} ∪ set_of is

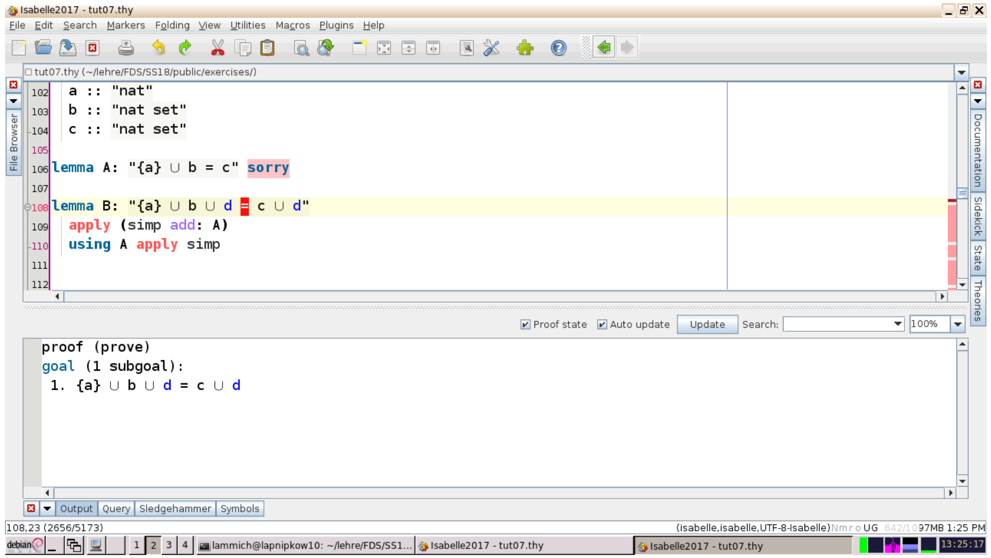

Isabelle2017 - tut07.thy
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
qed
lemma add_correct:
assumes "inv is"
shows "inv (add x is)" "set_of (add x is) = {x} ∪ set_of is"
using add_pres_inv assms tut07.inv_def apply fastforce
using assms set_of_add tut07.inv_def by fastforce
text ‹Hints:
Sketch the different cases (position of element relative to the first interval of the list)
theorem add_correct:
inv ?is ⟹ inv (add ?x ?is)
inv ?is ⟹ set_of (add ?x ?is) = {?x} ∪ set_of ?is

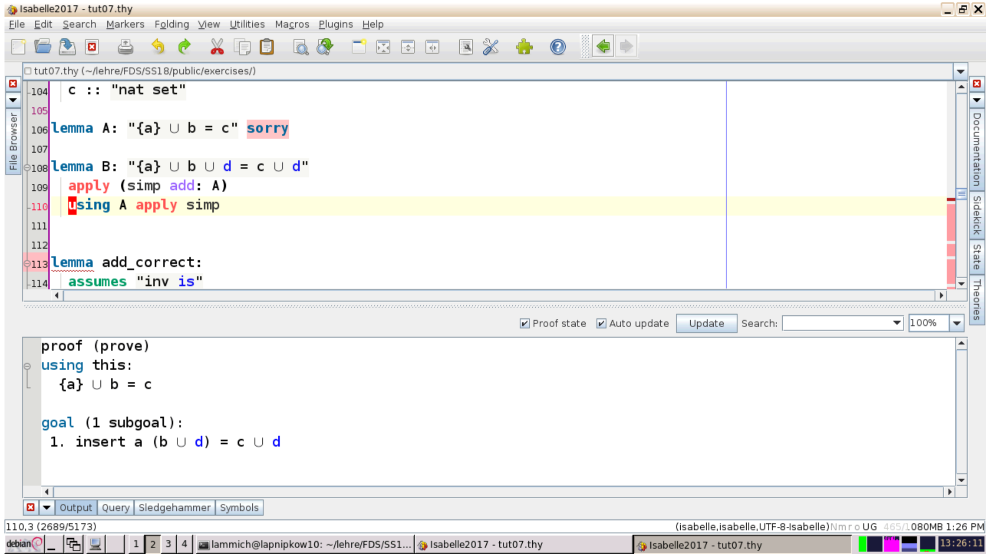

Isabelle2017 - tut07.thy
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
apply (cases ivs)
apply (auto split: if_splits)
done
qed
lemma add_correct:
assumes "inv is"
shows "inv (add x is)" "set_of (add x is) = {x} ∪ set_of is"
using add_pres_inv assms tut07.inv_def apply fastforce
using assms set_of_add tut07.inv_def by fastforce
proof (prove)
goal (1 subgoal):
1. set_of (add x is) = {x} ∪ set_of is

Isabelle2017 - tut07.thy
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
a :: "nat"
b :: "nat set"
c :: "nat set"
lemma A: "{a} ∪ b = c" sorry
lemma B: "{a} ∪ b ∪ d = c ∪ d"
lemma add_correct:
assumes "inv is"
shows "inv (add x is)" "set_of (add x is) = {x} ∪ set_of is"
proof (prove)
goal (1 subgoal):
1. {a} ∪ b ∪ d = c ∪ d

Isabelle2017 - tut07.thy
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
consts
a :: "nat"
b :: "nat set"
c :: "nat set"
lemma A: "{a} ∪ b = c" sorry
lemma B: "{a} ∪ b ∪ d = c ∪ d"
lemma add_correct:
proof (prove)
goal (1 subgoal):
1. {a} ∪ b ∪ d = c ∪ d

Isabelle2017 - tut07.thy (modified)
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
consts
a :: "nat"
b :: "nat set"
c :: "nat set"
lemma A: "{a} ∪ b = c" sorry
lemma B: "{a} ∪ b ∪ d = c ∪ d"
apply (simp add: A)
lemma add_correct:
proof (prove)
goal (1 subgoal):
1. insert a (b ∪ d) = c ∪ d
Input/output complete

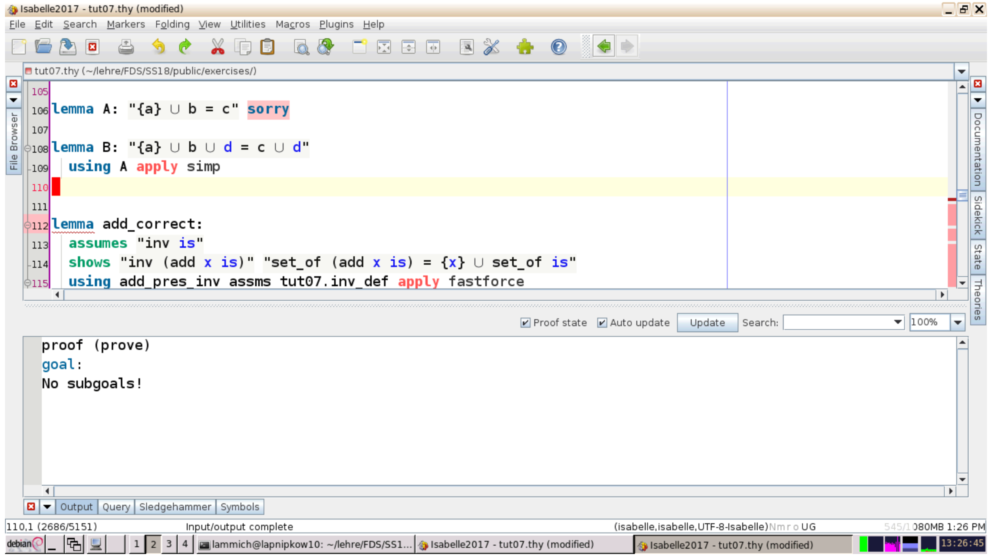

Isabelle2017 - tut07.thy (modified)
a :: "nat"
b :: "nat set"
c :: "nat set"
lemma A: "{a} ∪ b = c" sorry
lemma B: "{a} ∪ b ∪ d = c ∪ d"
apply (simp add: A)
using A apply simp
lemma add_correct:
assumes "inv is"
proof (prove)
goal (1 subgoal):
1. insert a (b ∪ d) = c ∪ d

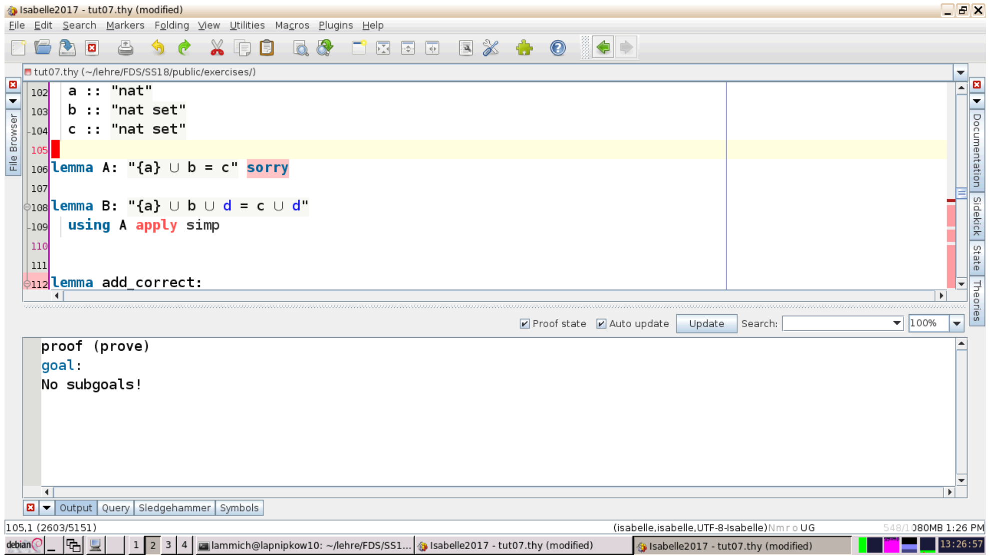

Isabelle2017 - tut07.thy (modified)
a :: "nat"
b :: "nat set"
c :: "nat set"
lemma A: "{a} ∪ b = c" sorry
lemma B: "{a} ∪ b ∪ d = c ∪ d"
apply (simp add: A)
using A apply simp
lemma add_correct:
assumes "inv is"
theorem A: {a} ∪ b = c

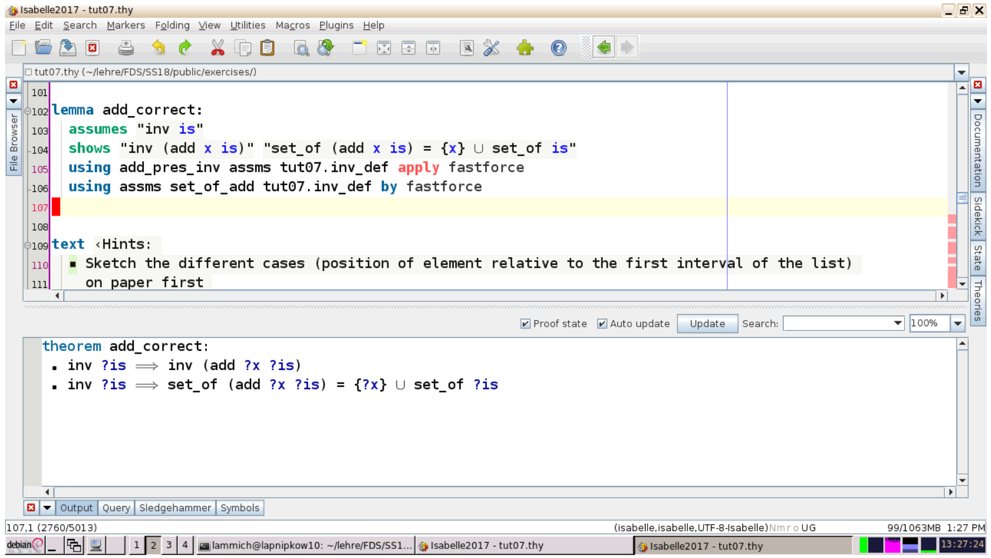

Isabelle2017 - tut07.thy
a :: "nat"
b :: "nat set"
c :: "nat set"
lemma A: "{a} ∪ b = c" sorry
lemma B: "{a} ∪ b ∪ d = c ∪ d"
apply (simp add: A)
using A apply simp
lemma add_correct:
assumes "inv is"
proof (prove)
using this:
{a} ∪ b = c
goal (1 subgoal):

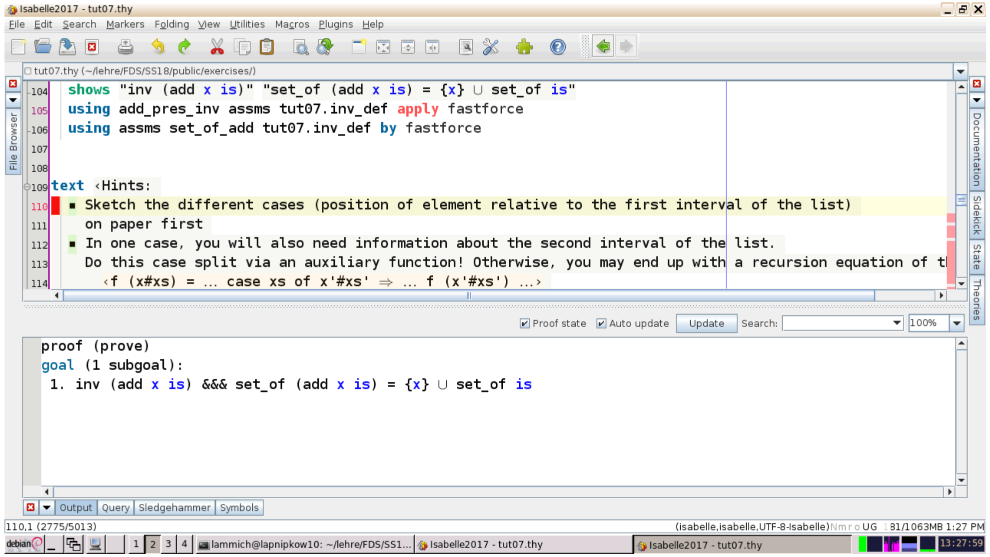

Isabelle2017 - tut07.thy
a :: "nat"
b :: "nat set"
c :: "nat set"
lemma A: "{a} ∪ b = c" sorry
lemma B: "{a} ∪ b ∪ d = c ∪ d"
apply (simp add: A)
using A apply simp
proof (prove)
goal (1 subgoal):
1. {a} ∪ b ∪ d = c ∪ d

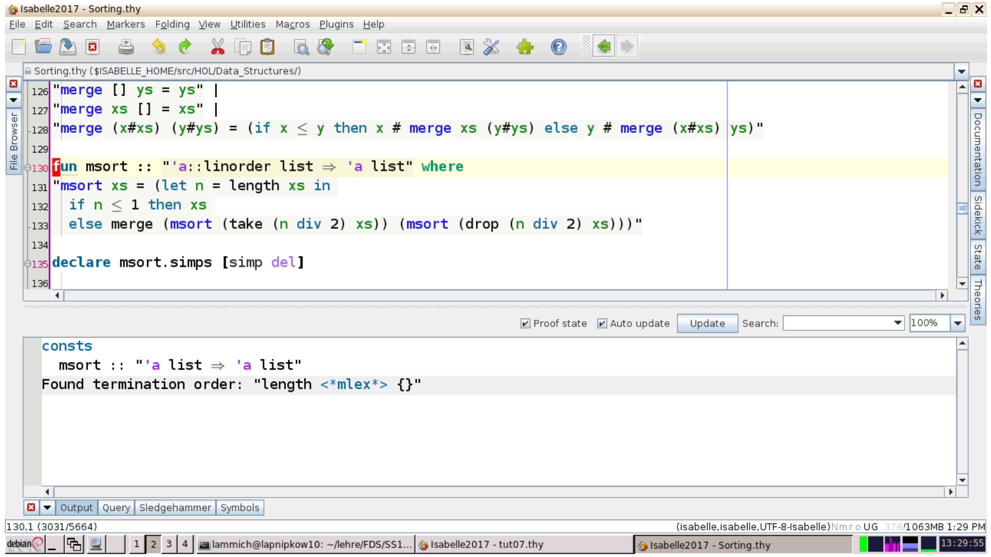

Isabelle2017 - tut07.thy
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
c :: "nat set"
lemma A: "{a} ∪ b = c" sorry
lemma B: "{a} ∪ b ∪ d = c ∪ d"
apply (simp add: A)
using A apply simp
lemma add_correct:
assumes "inv is"
proof (prove)
using this:
{a} ∪ b = c
goal (1 subgoal):
1. insert a (b ∪ d) = c ∪ d

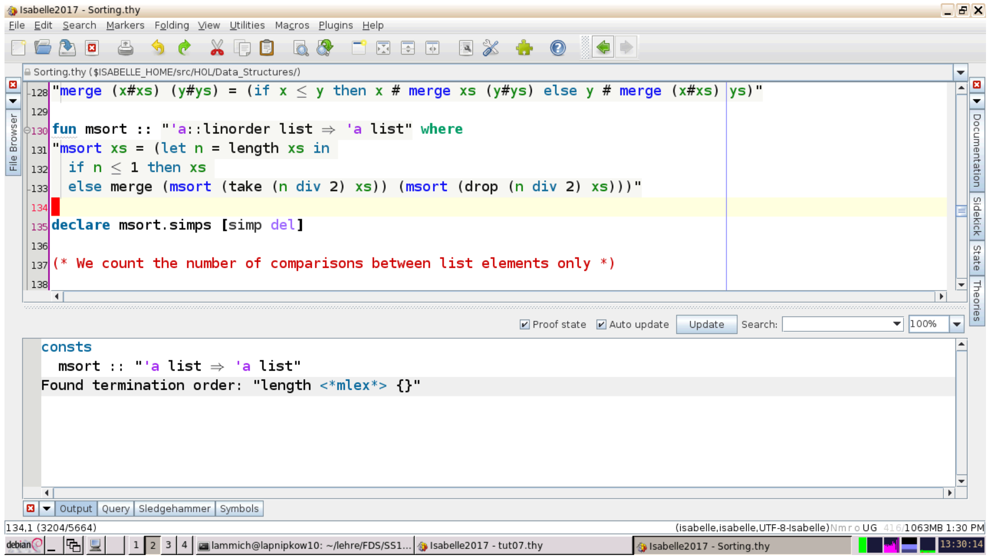

Isabelle2017 - tut07.thy (modified)
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
lemma A: "{a} ∪ b = c" sorry
lemma B: "{a} ∪ b ∪ d = c ∪ d"
using A apply simp
lemma add_correct:
assumes "inv is"
shows "inv (add x is)" "set_of (add x is) = {x} ∪ set_of is"
using add_pres_inv assms tut07.inv_def apply fastforce
proof (prove)
goal:
No subgoals!
Input/output complete

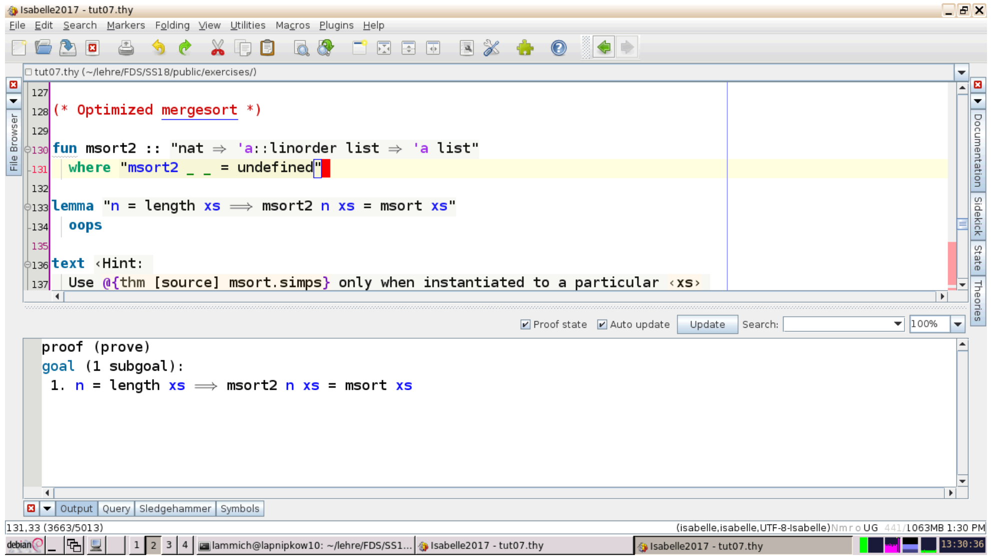

Isabelle2017 - tut07.thy (modified)
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
a :: "nat"
b :: "nat set"
c :: "nat set"
lemma A: "{a} ∪ b = c" sorry
lemma B: "{a} ∪ b ∪ d = c ∪ d"
using A apply simp
lemma add_correct:
proof (prove)
goal:
No subgoals!

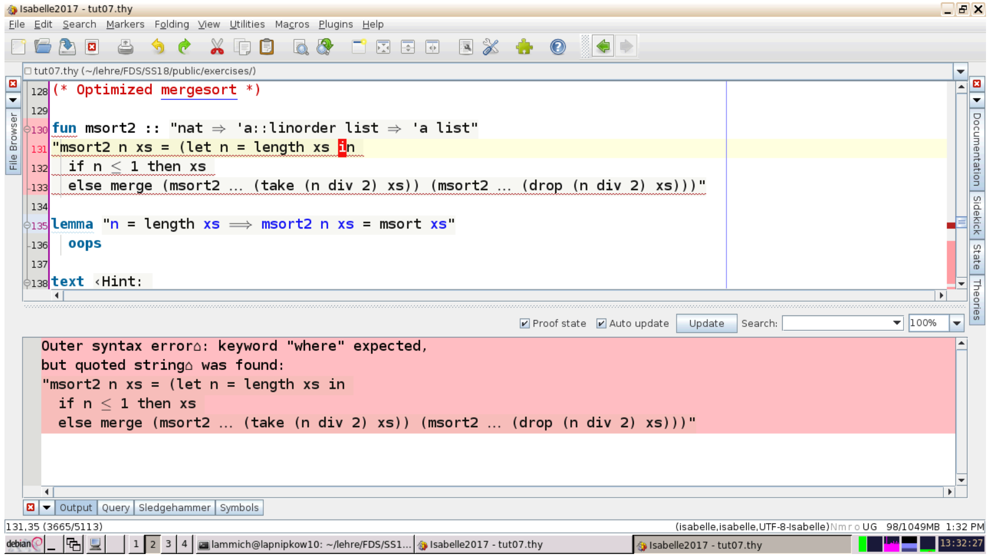

Isabelle2017 - tut07.thy
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
lemma add_correct:
assumes "inv is"
shows "inv (add x is)" "set_of (add x is) = {x} ∪ set_of is"
using add_pres_inv assms tut07.inv_def apply fastforce
using assms set_of_add tut07.inv_def by fastforce
text ‹Hints:
▪ Sketch the different cases (position of element relative to the first interval of the list)
on paper first
theorem add_correct:
• inv ?is ⟹ inv (add ?x ?is)
• inv ?is ⟹ set_of (add ?x ?is) = {?x} ∪ set_of ?is

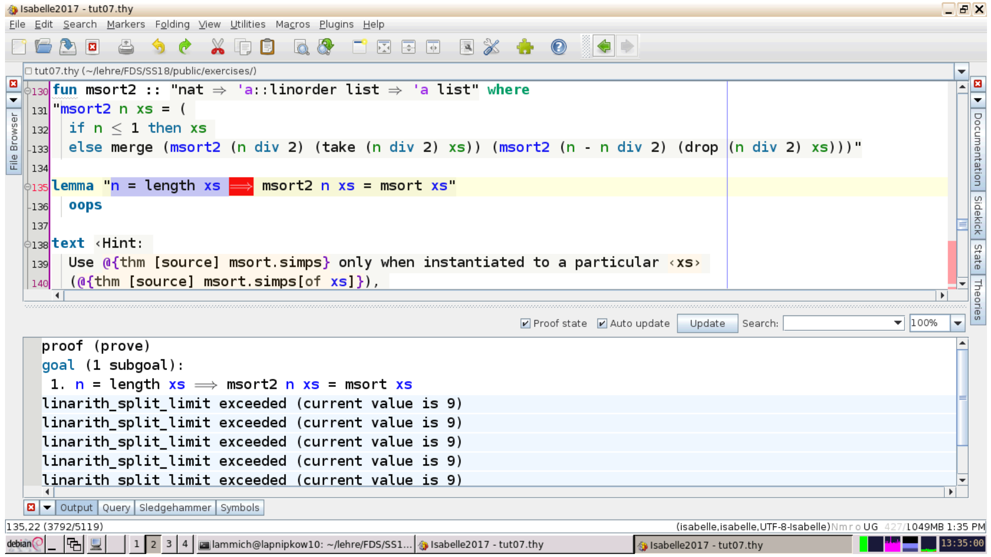

Isabelle2017 - tut07.thy
shows "inv (add x is)" "set_of (add x is) = {x} ∪ set_of is"
using add_pres_inv assms tut07.inv_def apply fastforce
using assms set_of_add tut07.inv_def by fastforce
text ‹Hints:
▪ Sketch the different cases (position of element relative to the first interval of the list)
on paper first
▪ In one case, you will also need information about the second interval of the list.
Do this case split via an auxiliary function! Otherwise, you may end up with a recursion equation of t
‹f (x#xs) = … case xs of x'#xs' ⇒ … f (x'#xs') …›
proof (prove)
goal (1 subgoal):
1. inv (add x is) &&& set_of (add x is) = {x} ∪ set_of is

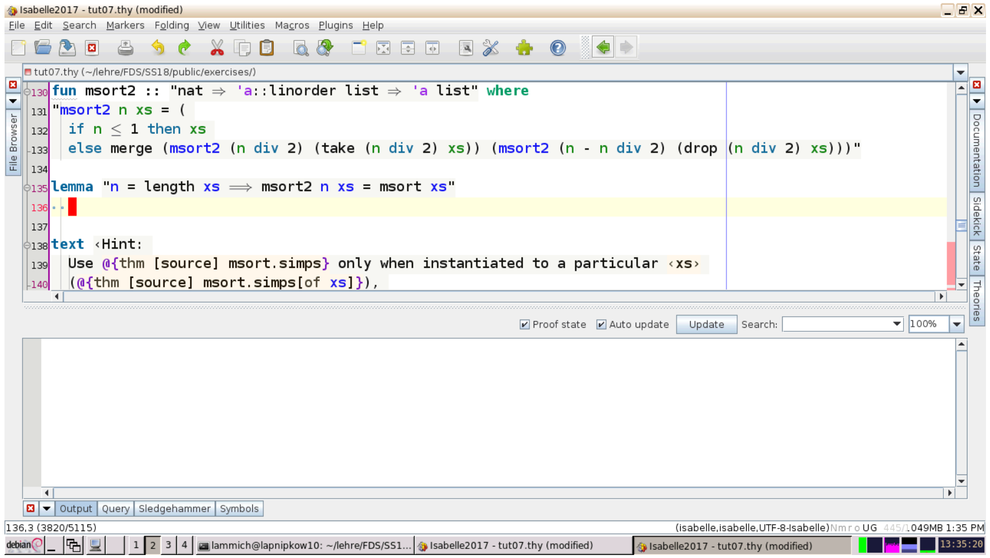

Isabelle2017 - Sorting.thy
"merge [] ys = ys" |
"merge xs [] = xs" |
"merge (x#xs) (y#ys) = (if x ≤ y then x # merge xs (y#ys) else y # merge (x#xs) ys)"
fun msort :: "'a::linorder list ⇒ 'a list" where
"msort xs = (let n = length xs in
if n ≤ 1 then xs
else merge (msort (take (n div 2) xs)) (msort (drop (n div 2) xs)))"
declare msort.simps [simp del]
consts
msort :: "'a list ⇒ 'a list"
Found termination order: "length <*mlex*> {}"

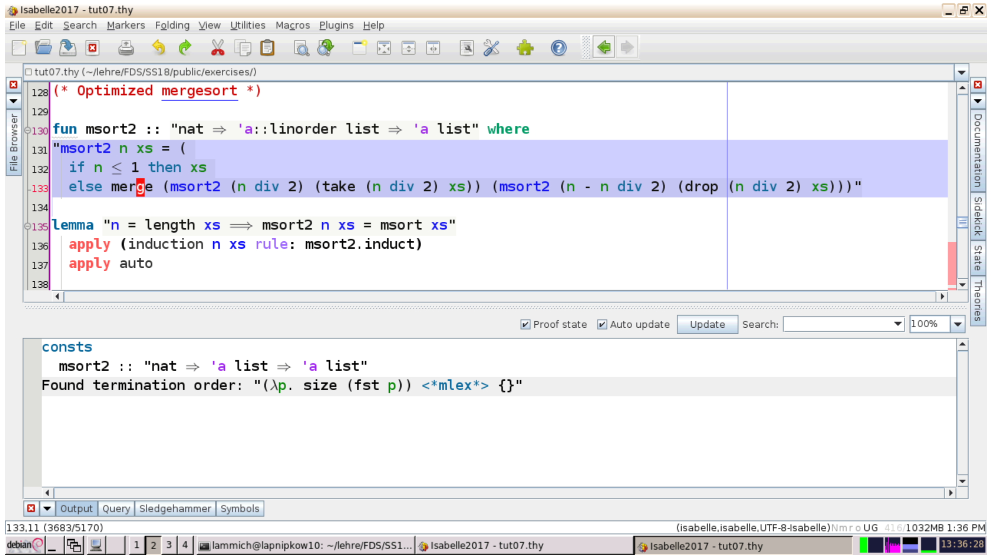

Isabelle2017 - Sorting.thy
"merge (x#xs) (y#ys) = (if x ≤ y then x # merge xs (y#ys) else y # merge (x#xs) ys)"
fun msort :: "'a::linorder list ⇒ 'a list" where
"msort xs = (let n = length xs in
if n ≤ 1 then xs
else merge (msort (take (n div 2) xs)) (msort (drop (n div 2) xs)))"
declare msort.simps [simp del]
(* We count the number of comparisons between list elements only *)
consts
msort :: "'a list ⇒ 'a list"
Found termination order: "length <*mlex*> {}"

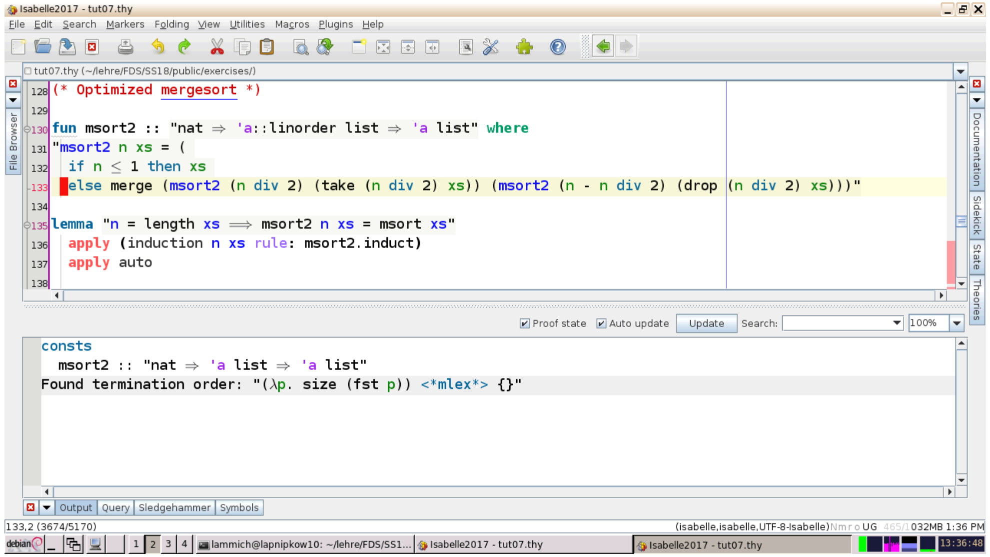

Isabelle2017 - tut07.thy
(* Optimized mergesort *)
fun msort2 :: "nat ⇒ 'a::linorder list ⇒ 'a list"
where "msort2 _ _ = undefined"
lemma "n = length xs ⟹ msort2 n xs = msort xs"
oops
text ‹Hint:
Use @{thm [source] msort.simps} only when instantiated to a particular ‹xs›
proof (prove)
goal (1 subgoal):
1. n = length xs ⟹ msort2 n xs = msort xs

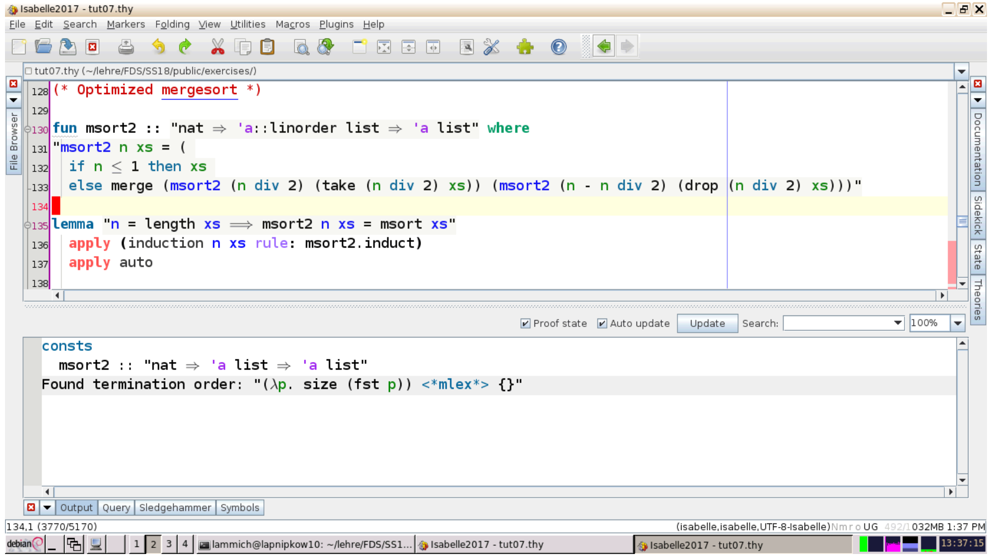

Isabelle2017 - tut07.thy
(* Optimized mergesort *)
fun msort2 :: "nat ⇒ 'a::linorder list ⇒ 'a list"
"msort2 n xs = (let n = length xs in
if n ≤ 1 then xs
else merge (msort2 … (take (n div 2) xs)) (msort2 … (drop (n div 2) xs)))"
lemma "n = length xs ⟹ msort2 n xs = msort xs"
oops
text ‹Hint:
Outer syntax error⌂: keyword "where" expected,
but quoted string⌂ was found:
"msort2 n xs = (let n = length xs in
if n ≤ 1 then xs
else merge (msort2 … (take (n div 2) xs)) (msort2 … (drop (n div 2) xs)))"

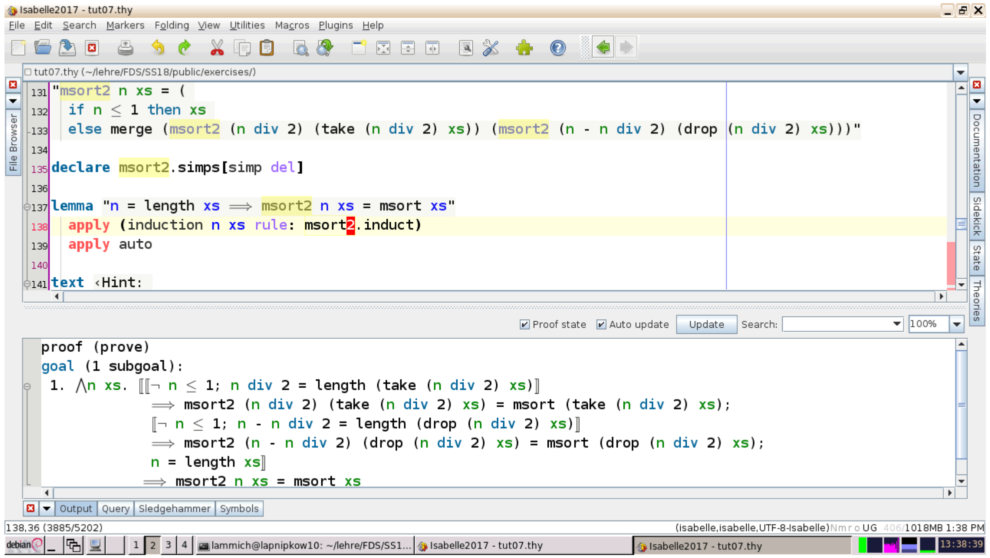

Isabelle2017 - tut07.thy
fun msort2 :: "nat ⇒ 'a::linorder list ⇒ 'a list" where
"msort2 n xs = (
if n ≤ 1 then xs
else merge (msort2 (n div 2) (take (n div 2) xs)) (msort2 (n - n div 2) (drop (n div 2) xs)))"
lemma "n = length xs ⟹ msort2 n xs = msort xs"
oops
text ‹Hint:
Use @{thm [source] msort.simps} only when instantiated to a particular ‹xs›
(@{thm [source] msort.simps[of xs]}),
proof (prove)
goal (1 subgoal):
1. n = length xs ⟹ msort2 n xs = msort xs
linarith_split_limit exceeded (current value is 9)
linarith_split_limit exceeded (current value is 9)
linarith_split_limit exceeded (current value is 9)
linarith_split_limit exceeded (current value is 9)
linarith_split_limit exceeded (current value is 9)

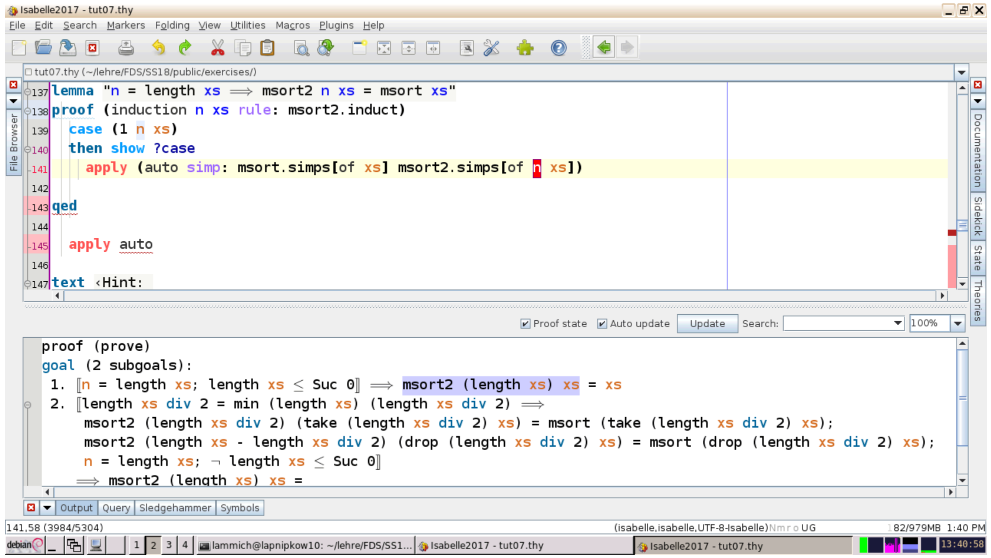

Isabelle2017 - tut07.thy (modified)
fun msort2 :: "nat ⇒ 'a::linorder list ⇒ 'a list" where
"msort2 n xs = (
if n ≤ 1 then xs
else merge (msort2 (n div 2) (take (n div 2) xs)) (msort2 (n - n div 2) (drop (n div 2) xs)))"
lemma "n = length xs ⟹ msort2 n xs = msort xs"
text ‹Hint:
Use @{thm [source] msort.simps} only when instantiated to a particular ‹xs›
(@{thm [source] msort.simps[of xs]}),

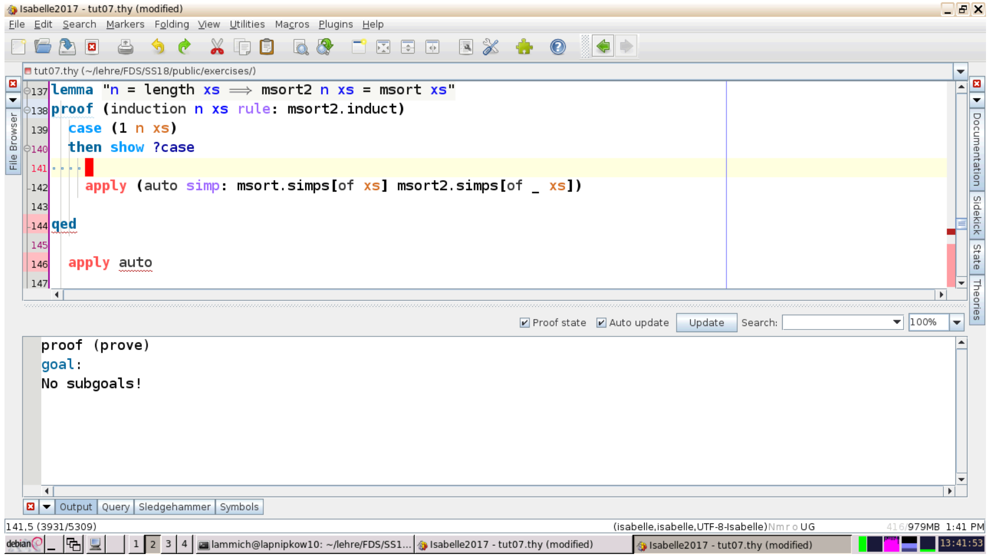

Isabelle2017 - tut07.thy
(* Optimized mergesort *)
fun msort2 :: "nat ⇒ 'a::linorder list ⇒ 'a list" where
"msort2 n xs = (
if n ≤ 1 then xs
else merge (msort2 (n div 2) (take (n div 2) xs)) (msort2 (n - n div 2) (drop (n div 2) xs)))"
lemma "n = length xs ⟹ msort2 n xs = msort xs"
apply (induction n xs rule: msort2.induct)
apply auto
consts
msort2 :: "nat ⇒ 'a list ⇒ 'a list"
Found termination order: "(λp. size (fst p)) <*mlex*> {}"

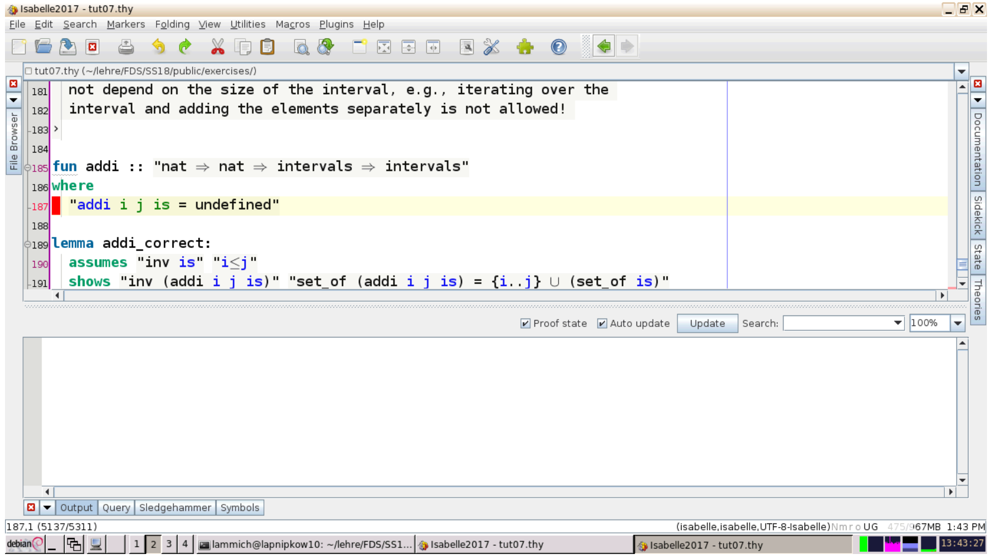

Isabelle2017 - tut07.thy
(* Optimized mergesort *)
fun msort2 :: "nat ⇒ 'a::linorder list ⇒ 'a list" where
"msort2 n xs = (
if n ≤ 1 then xs
else merge (msort2 (n div 2) (take (n div 2) xs)) (msort2 (n - n div 2) (drop (n div 2) xs)))"
lemma "n = length xs ⟹ msort2 n xs = msort xs"
apply (induction n xs rule: msort2.induct)
apply auto
consts
msort2 :: "nat ⇒ 'a list ⇒ 'a list"
Found termination order: "(λp. size (fst p)) <*mlex*> {}"

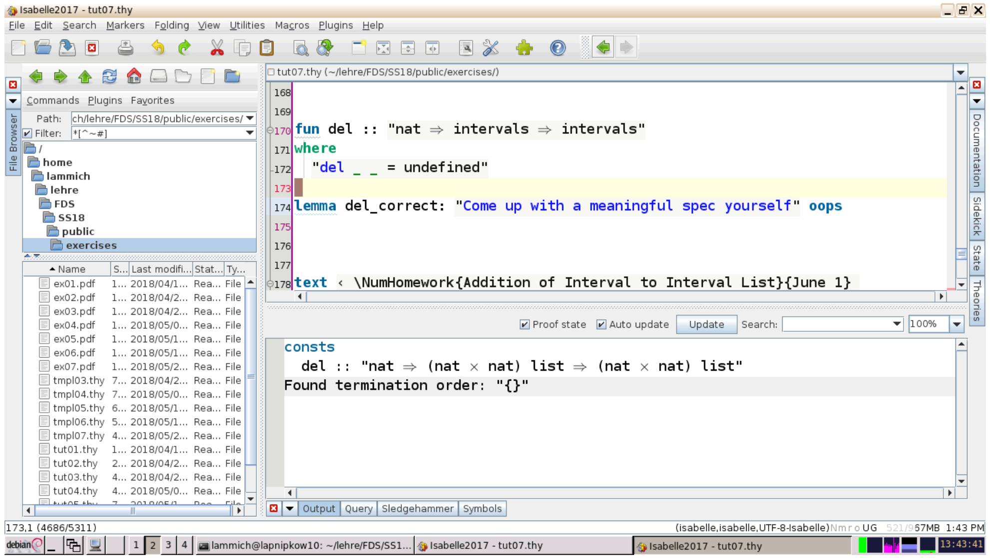

Isabelle2017 - tut07.thy
(* Optimized mergesort *)
fun msort2 :: "nat ⇒ 'a::linorder list ⇒ 'a list" where
"msort2 n xs = (
if n ≤ 1 then xs
else merge (msort2 (n div 2) (take (n div 2) xs)) (msort2 (n - n div 2) (drop (n div 2) xs)))"
lemma "n = length xs ⟹ msort2 n xs = msort xs"
apply (induction n xs rule: msort2.induct)
apply auto
consts
msort2 :: "nat ⇒ 'a list ⇒ 'a list"
Found termination order: "(λp. size (fst p)) <*mlex*> {}"

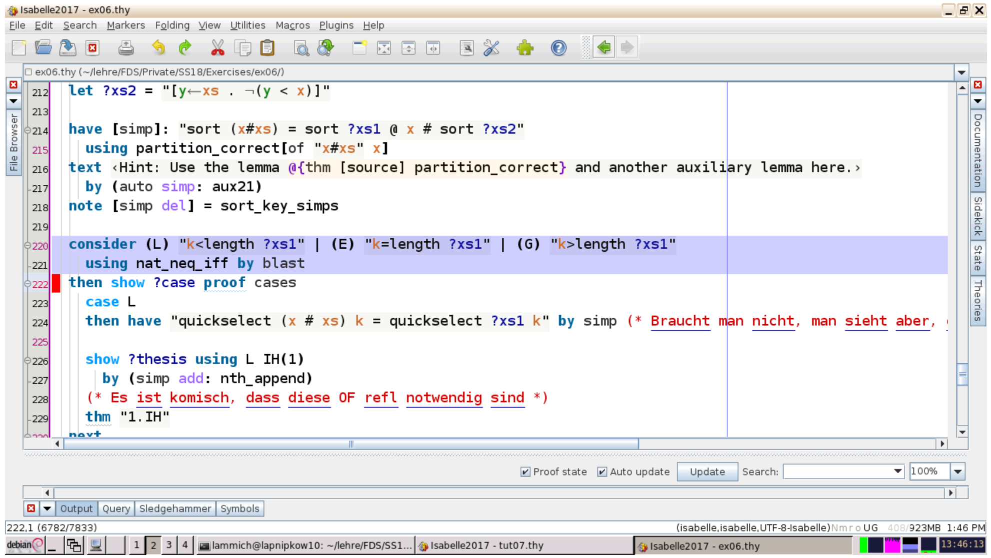

Isabelle2017 - tut07.thy
"msort2 n xs = (
if n ≤ 1 then xs
else merge (msort2 (n div 2) (take (n div 2) xs)) (msort2 (n - n div 2) (drop (n div 2) xs)))"
declare msort2.simps[simp del]
lemma "n = length xs ⟹ msort2 n xs = msort xs"
apply (induction n xs rule: msort2.induct)
apply auto
text ‹Hint:
proof (prove)
goal (1 subgoal):
1. ⋀n xs. ⟦¬ n ≤ 1; n div 2 = length (take (n div 2) xs)⟧
⟹ msort2 (n div 2) (take (n div 2) xs) = msort (take (n div 2) xs);
⟦¬ n ≤ 1; n - n div 2 = length (drop (n div 2) xs)⟧
⟹ msort2 (n - n div 2) (drop (n div 2) xs) = msort (drop (n div 2) xs);
n = length xs⟧
⟹ msort2 n xs = msort xs

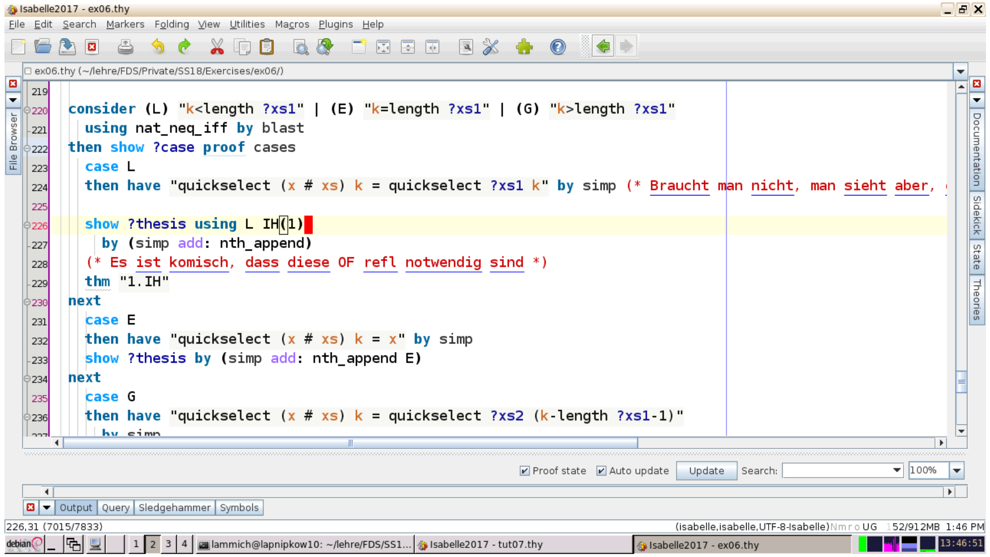

Isabelle2017 - tut07.thy
lemma "n = length xs ⟹ msort2 n xs = msort xs"
proof (induction n xs rule: msort2.induct)
case (1 n xs)
then show ?case
apply (auto simp: msort.simps[of xs] msort2.simps[of n xs])
qed
apply auto
text ‹Hint:
proof (prove)
goal (2 subgoals):
1. ⟦n = length xs; length xs ≤ Suc 0⟧ ⟹ msort2 (length xs) xs = xs
2. ⟦length xs div 2 = min (length xs) (length xs div 2) ⟹
msort2 (length xs div 2) (take (length xs div 2) xs) = msort (take (length xs div 2) xs);
msort2 (length xs - length xs div 2) (drop (length xs div 2) xs) = msort (drop (length xs div 2) xs);
n = length xs; ¬ length xs ≤ Suc 0⟧
⟹ msort2 (length xs) xs =

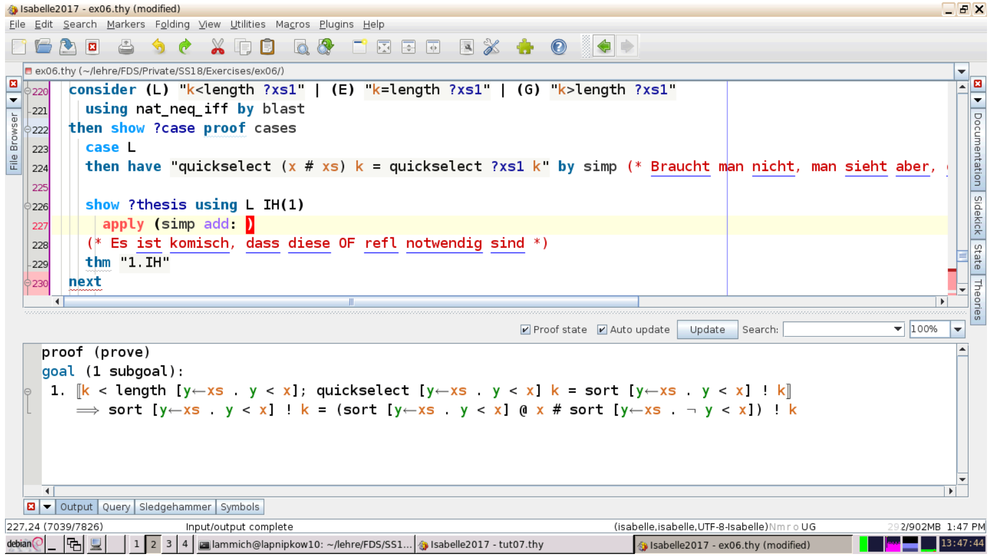

Isabelle2017 - tut07.thy (modified)
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
lemma "n = length xs ⟹ msort2 n xs = msort xs"
proof (induction n xs rule: msort2.induct)
case (1 n xs)
then show ?case
apply (auto simp: msort.simps[of xs] msort2.simps[of _ xs])
qed
apply auto
proof (prove)
goal:
No subgoals!
141,5 (3931/5309)

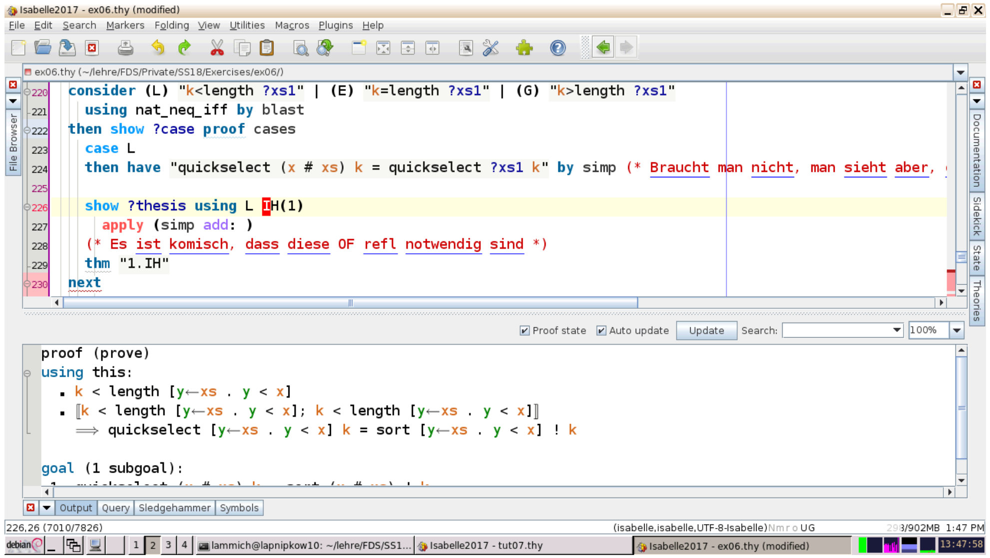

Isabelle2017 - tut07.thy
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
not depend on the size of the interval, e.g., iterating over the
interval and adding the elements separately is not allowed!
fun addi :: "nat ⇒ nat ⇒ intervals ⇒ intervals"
where
"addi i j is = undefined"
lemma addi_correct:
assumes "inv is" "i≤j"
shows "inv (addi i j is)" "set_of (addi i j is) = {i..j} ∪ (set_of is)"
187,1 (5137/5311)

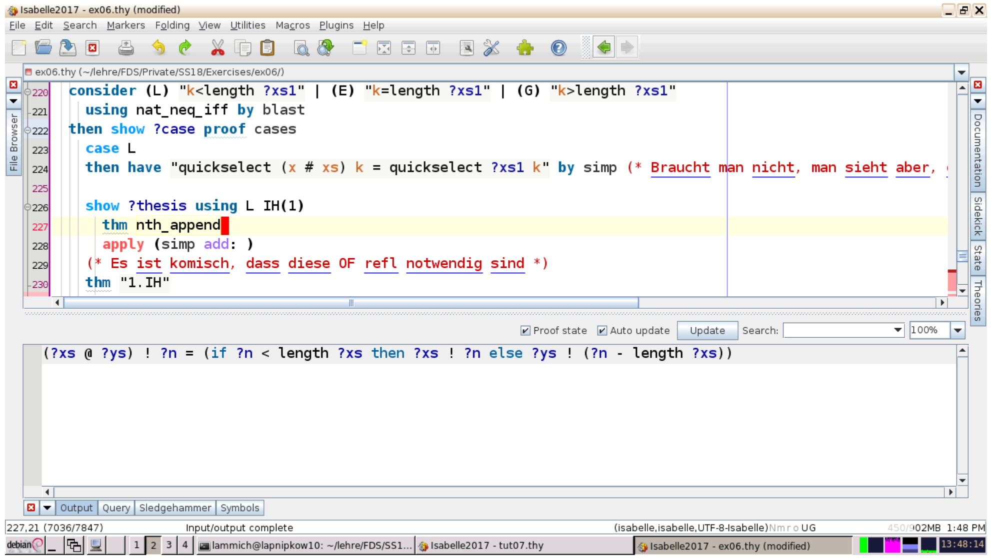

Isabelle2017 - tut07.thy
tut07.thy (~/lehre/FDS/SS18/public/exercises/)
fun del :: "nat ⇒ intervals ⇒ intervals"
where
"del _ _ = undefined"
lemma del_correct: "Come up with a meaningful spec yourself" oops
text ‹ \NumHomework{Addition of Interval to Interval List}{June 1}
consts
del :: "nat ⇒ (nat × nat) list ⇒ (nat × nat) list"
Found termination order: "{}"
173,1 (4686/5311)

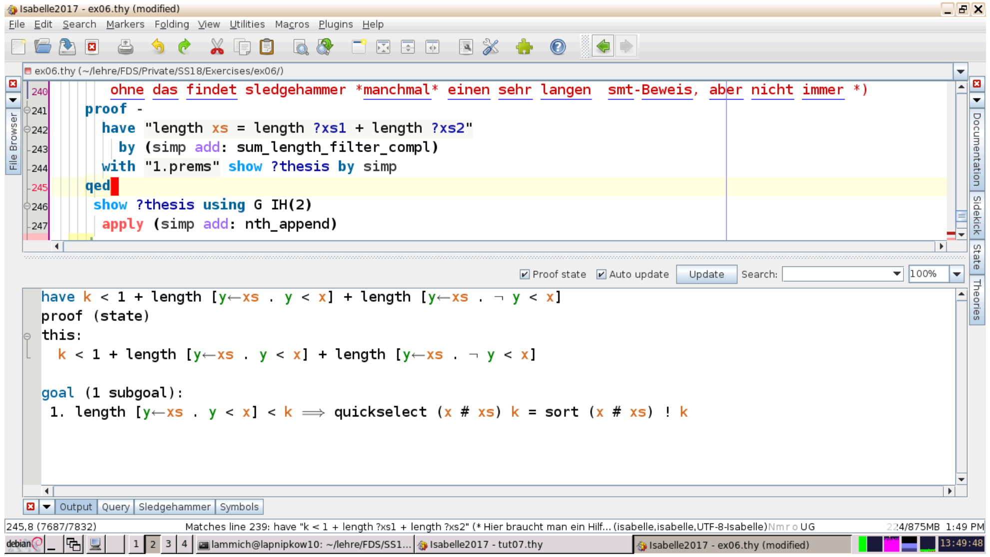

Isabelle2017 - ex06.thy
ex06.thy (~/lehre/FDS/Private/SS18/Exercises/ex06/)
let ?xs2 = "[y←xs . ¬(y < x)]"
have [simp]: "sort (x#xs) = sort ?xs1 @ x # sort ?xs2"
using partition_correct[of "x#xs" x]
text ‹Hint: Use the lemma @{thm [source] partition_correct} and another auxiliary lemma here.›
by (auto simp: aux21)
note [simp del] = sort_key_simps
consider (L) "k<length ?xs1" | (E) "k=length ?xs1" | (G) "k>length ?xs1"
using nat_neq_iff by blast
then show ?case proof cases
case L
then have "quickselect (x # xs) k = quickselect ?xs1 k" by simp (* Braucht man nicht, man sieht aber,
show ?thesis using L IH(1)
by (simp add: nth_append)
(* Es ist komisch, dass diese OF refl notwendig sind *)
thm "1.IH"
222,1 (6782/7833)

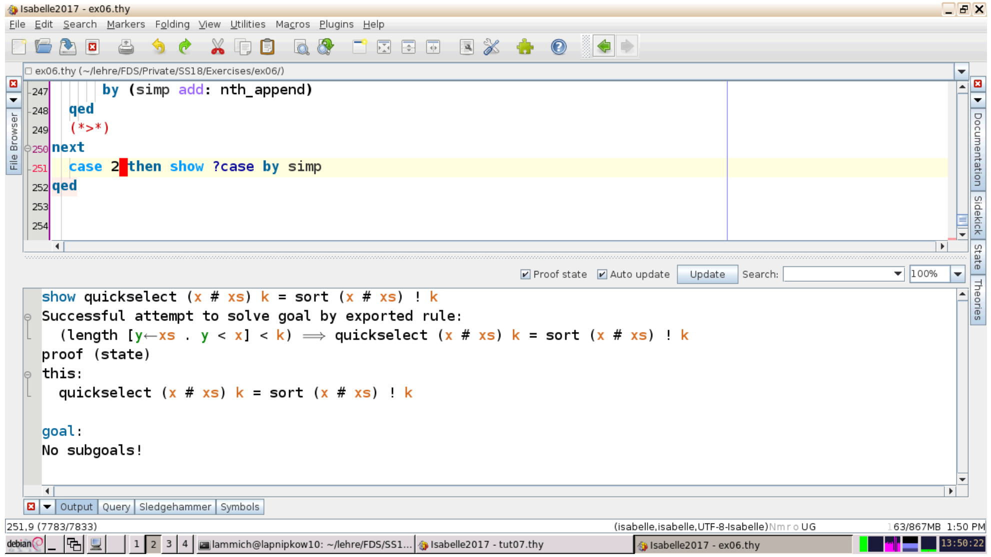

Isabelle2017 - ex06.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help
ex06.thy (~/lehre/FDS/Private/SS18/Exercises/ex06/)
consider (L) "k<length ?xs1" | (E) "k=length ?xs1" | (G) "k>length ?xs1"
using nat_neq_iff by blast
then show ?case proof cases
case L
then have "quickselect (x # xs) k = quickselect ?xs1 k" by simp (* Braucht man nicht, man sieht aber,
show ?thesis using L IH(1)
by (simp add: nth_append)
(* Es ist komisch, dass diese OF refl notwendig sind *)
thm "1.IH"
next
case E
then have "quickselect (x # xs) k = x" by simp
show ?thesis by (simp add: nth_append E)
next
case G
then have "quickselect (x # xs) k = quickselect ?xs2 (k-length ?xs1-1)"
Proof state
Auto update
Update
Search:
100%
Output
Query
Sledgehammer
Symbols
226,31 (7015/7833)
13:46:51

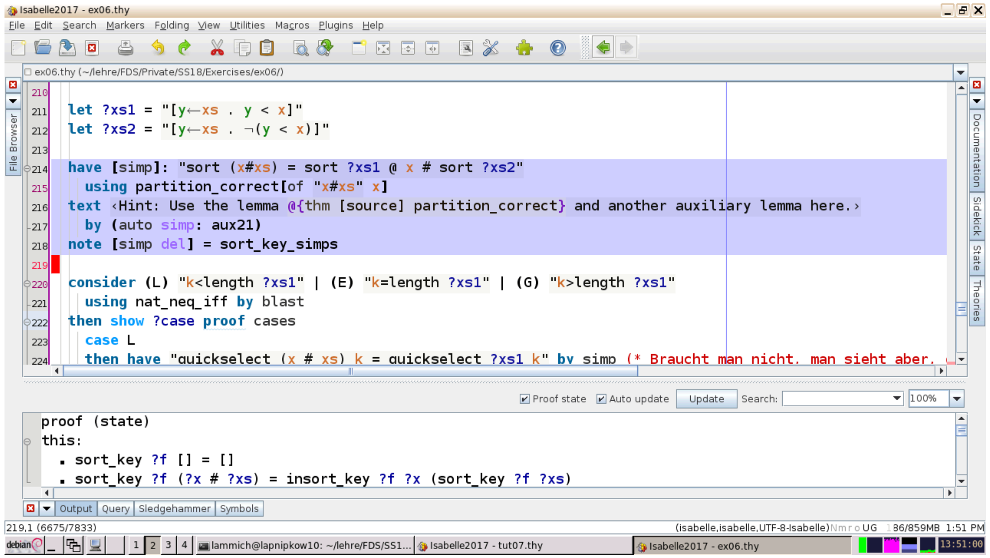

Isabelle2017 - ex06.thy (modified)
consider (L) "k<length ?xs1" | (E) "k=length ?xs1" | (G) "k>length ?xs1"
using nat_neq_iff by blast
then show ?case proof cases
case L
then have "quickselect (x # xs) k = quickselect ?xs1 k" by simp (* Braucht man nicht, man sieht aber,
show ?thesis using L IH(1)
apply (simp add:)
(* Es ist komisch, dass diese OF refl notwendig sind *)
thm "1.IH"
next
proof (prove)
goal (1 subgoal):
1. ⟦k < length [y←xs . y < x]; quickselect [y←xs . y < x] k = sort [y←xs . y < x] ! k⟧
⟹ sort [y←xs . y < x] ! k = (sort [y←xs . y < x] @ x # sort [y←xs . ¬ y < x]) ! k
227,24 (7039/7826)
Input/output complete
13:47:44

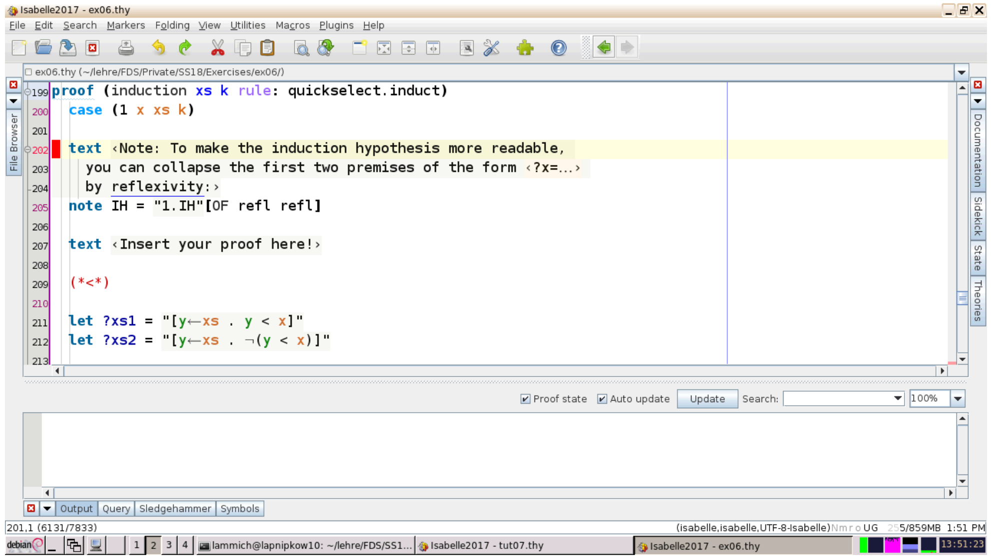

Isabelle2017 - ex06.thy (modified)
consider (L) "k<length ?xs1" | (E) "k=length ?xs1" | (G) "k>length ?xs1"
using nat_neq_iff by blast
then show ?case proof cases
case L
then have "quickselect (x # xs) k = quickselect ?xs1 k" by simp (* Braucht man nicht, man sieht aber,
show ?thesis using L IH(1)
apply (simp add:)
(* Es ist komisch, dass diese OF refl notwendig sind *)
thm "1.IH"
next
proof (prove)
using this:
k < length [y←xs . y < x]
⟦k < length [y←xs . y < x]; k < length [y←xs . y < x]⟧
⟹ quickselect [y←xs . y < x] k = sort [y←xs . y < x] ! k
goal (1 subgoal):
226,26 (7010/7826)
13:47:58

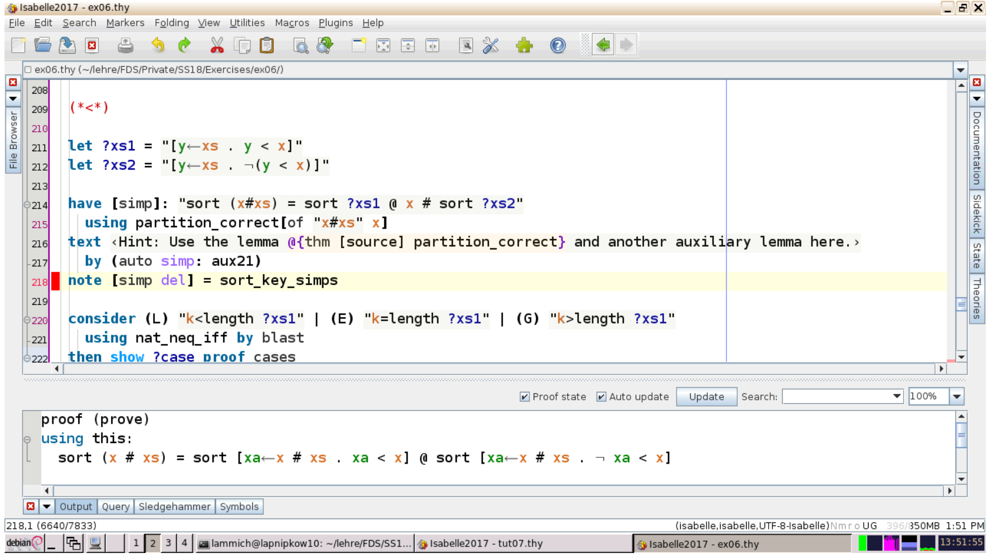

Isabelle2017 - ex06.thy (modified)
consider (L) "k<length ?xs1" | (E) "k=length ?xs1" | (G) "k>length ?xs1"
using nat_neq_iff by blast
then show ?case proof cases
case L
then have "quickselect (x # xs) k = quickselect ?xs1 k" by simp (* Braucht man nicht, man sieht aber,
show ?thesis using L IH(1)
thm nth_append
apply (simp add:)
(* Es ist komisch, dass diese OF refl notwendig sind *)
thm "1.IH"
(?xs @ ?ys) ! ?n = (if ?n < length ?xs then ?xs ! ?n else ?ys ! (?n - length ?xs))
227,21 (7036/7847)
Input/output complete
13:48:14


Isabelle2017 - ex06.thy (modified)
ohne das findet sledgehammer *manchmal* einen sehr langen smt-Beweis, aber nicht immer *)
proof -
have "length xs = length ?xs1 + length ?xs2"
by (simp add: sum_length_filter_compl)
with "1.prems" show ?thesis by simp
qed
show ?thesis using G IH(2)
apply (simp add: nth_append)
have k < 1 + length [y←xs . y < x] + length [y←xs . ¬ y < x]
proof (state)
this:
k < 1 + length [y←xs . y < x] + length [y←xs . ¬ y < x]
goal (1 subgoal):
1. length [y←xs . y < x] < k ⟹ quickselect (x # xs) k = sort (x # xs) ! k


Isabelle2017 - ex06.thy
by (simp add: nth_append)
qed
(*>*)
next
case 2 then show ?case by simp
qed
show quickselect (x # xs) k = sort (x # xs) ! k
Successful attempt to solve goal by exported rule:
(length [y←xs . y < x] < k) ⟹ quickselect (x # xs) k = sort (x # xs) ! k
proof (state)
this:
quickselect (x # xs) k = sort (x # xs) ! k
goal:
No subgoals!


Isabelle2017 - ex06.thy
let ?xs1 = "[y←xs . y < x]"
let ?xs2 = "[y←xs . ¬(y < x)]"
have [simp]: "sort (x#xs) = sort ?xs1 @ x # sort ?xs2"
using partition_correct[of "x#xs" x]
text ‹Hint: Use the lemma @{thm [source] partition_correct} and another auxiliary lemma here.›
by (auto simp: aux21)
note [simp del] = sort_key_simps
consider (L) "k<length ?xs1" | (E) "k=length ?xs1" | (G) "k>length ?xs1"
using nat_neq_iff by blast
then show ?case proof cases
case L
then have "quickselect (x # xs) k = quickselect ?xs1 k" by simp (* Braucht man nicht, man sieht aber,
proof (state)
this:
sort_key ?f [] = []
sort_key ?f (?x # ?xs) = insort_key ?f ?x (sort_key ?f ?xs)


Isabelle2017 - ex06.thy
proof (induction xs k rule: quickselect.induct)
case (1 x xs k)
text ‹Note: To make the induction hypothesis more readable,
you can collapse the first two premises of the form ‹?x=...›
by reflexivity:›
note IH = "1.IH"[OF refl refl]
text ‹Insert your proof here!›
(*<*)
let ?xs1 = "[y←xs . y < x]"
let ?xs2 = "[y←xs . ¬(y < x)]"


Isabelle2017 - ex06.thy
File Edit Search Markers Folding View Utilities Macros Plugins Help
ex06.thy (~/lehre/FDS/Private/SS18/Exercises/ex06/)
(*<*)
let ?xs1 = "[y←xs . y < x]"
let ?xs2 = "[y←xs . ¬(y < x)]"
have [simp]: "sort (x#xs) = sort ?xs1 @ x # sort ?xs2"
using partition_correct[of "x#xs" x]
text ‹Hint: Use the lemma @{thm [source] partition_correct} and another auxiliary lemma here.›
by (auto simp: aux21)
note [simp del] = sort_key_simps
consider (L) "k<length ?xs1" | (E) "k=length ?xs1" | (G) "k>length ?xs1"
using nat_neq_iff by blast
then show ?case proof cases
Proof state
Auto update
Update
Search:
100%
proof (prove)
using this:
sort (x # xs) = sort [xa←x # xs . xa < x] @ sort [xa←x # xs . ¬ xa < x]
Output
Query
Sledgehammer
Symbols
218,1 (6640/7833)
(isabelle,isabelle,UTF-8-Isabelle)
1:51 PM
lammich@lapnipkow10: ~/lehre/FDS/SS1...
Isabelle2017 - tut07.thy
Isabelle2017 - ex06.thy
13:51:55