1 Frequently Used Auxiliary Lemmas

theory *AuxBox* = *PCCFramework*:

lemmas *set-mem-eq* = *mem-iff*

— changes the preprocessor for the simplifier, such that internally generated lemmas are augmented with proof objects
setup ⟪ [*Simplifier.change-simpset-of op setmksimps* (*setmp proofs 2* (*mksimps mksimps-pairs*))] ⟫

lemma *forall-switch2*:
$(\forall\ x\ y.\ P\ x\ y) \Longrightarrow (\forall\ y\ x.\ P\ x\ y)$

lemma *forall-switch3*:
$(\forall\ z.\ (\forall\ x\ .\ (\forall\ y.\ P\ x\ y\ z))) \Longrightarrow (\forall\ x\ y\ z.\ P\ x\ y\ z)$

lemma *forall-switch4*: $(\forall z\ w\ x\ y.\ P\ w\ x\ y\ z) \Longrightarrow (\forall w\ x\ y\ z.\ P\ w\ x\ y\ z)$

lemma *forall-switch5*:
$\forall\ z\ v\ w\ x\ y.\ P\ v\ w\ x\ y\ z \Longrightarrow \forall\ v.\ \forall\ w.\ \forall\ x.\ \forall\ y.\ \forall\ z.\ P\ v\ w\ x\ y\ z$

lemma *forall-switch6*:
$\forall\ y\ q\ z\ v\ w\ x.\ P\ v\ w\ x\ y\ z\ q \Longrightarrow \forall\ q.\ \forall\ z.\ \forall\ v.\ \forall\ w.\ \forall\ x.\ \forall\ y.\ P\ v\ w\ x\ y\ z\ q$

lemma *forall-switch7*:
$\forall\ r\ v\ w\ x\ y\ z\ q.\ P\ v\ w\ x\ y\ z\ q\ r \Longrightarrow \forall\ v.\ \forall\ w.\ \forall\ x.\ \forall\ y.\ \forall\ z.\ \forall\ q.\ \forall\ r.\ P\ v\ w\ x\ y\ z\ q\ r$

lemma *forall-switch8*:
$\forall\ r\ v\ w\ x\ y\ z\ q\ s.\ P\ v\ w\ x\ y\ z\ q\ s\ r \Longrightarrow \forall\ v.\ \forall\ w.\ \forall\ x.\ \forall\ y.\ \forall\ z.\ \forall\ q.\ \forall\ s.\ \forall\ r.\ P\ v\ w\ x\ y\ z\ q\ s\ r$

lemma *forall-switch9*:
$\forall\ z\ r\ s\ t\ u\ v\ w\ x\ y\ .\ P\ r\ s\ t\ u\ v\ w\ x\ y\ z \Longrightarrow \forall\ r.\ \forall\ s.\ \forall\ t.\ \forall\ u.\ \forall\ v.\ \forall\ w.\ \forall\ x.\ \forall\ y.\ \forall\ z.\ \ P\ r\ s\ t\ u\ v\ w\ x\ y\ z$

lemma *forall-switch10*:
$\forall\ z\ q\ r\ s\ t\ u\ v\ w\ x\ y\ .\ P\ q\ r\ s\ t\ u\ v\ w\ x\ y\ z \Longrightarrow \forall\ q.\ \forall\ r.\ \forall\ s.\ \forall\ t.\ \forall\ u.\ \forall\ v.\ \forall\ w.\ \forall\ x.\ \forall\ y.\ \forall\ z.\ \ P\ q\ r\ s\ t\ u\ v\ w\ x\ y\ z$

lemma *forall-switch11*:
$\forall\ z\ p\ q\ r\ s\ t\ u\ v\ w\ x\ y\ .\ P\ p\ q\ r\ s\ t\ u\ v\ w\ x\ y\ z \Longrightarrow \forall\ p.\ \forall\ q.\ \forall\ r.\ \forall\ s.\ \forall\ t.\ \forall\ u.\ \forall\ v.\ \forall\ w.\ \forall\ x.\ \forall\ y.\ \forall\ z.\ \ P\ p\ q\ r\ s\ t\ u\ v\ w\ x\ y\ z$

lemma *forall-switch12*:
$\forall\ z\ n\ p\ q\ r\ s\ t\ u\ v\ w\ x\ y.\ P\ n\ p\ q\ r\ s\ t\ u\ v\ w\ x\ y\ z \Longrightarrow \forall n.\ \forall\ p.\ \forall\ q.\ \forall\ r.\ \forall\ s.\ \forall\ t.\ \forall\ u.\ \forall\ v.\ \forall\ w.$

$\forall\ x.\ \forall\ y.\ \forall\ z.\ \ P\ n\ p\ q\ r\ s\ t\ u\ v\ w\ x\ y\ z$

lemma *notin-union-E*:
$x \notin (A \cup B) = ((x \notin A) \wedge (x \notin B))$

lemma *notin-subset*:
$[\![\ x \notin A;\ B \subseteq A\ ]\!] \Longrightarrow x \notin B$

lemma *notin-Union-iff*:
$A \notin (\bigcup\ S) = (\forall\ X{\in}S.\ A \notin X)$

lemma *refl-True*:
$(t = t) = True$

lemma *refl-Some*:
$(Some\ x = Some\ y) = (x = y)$

lemma *elem-set-append-cases*:
$\forall\ x\ l1\ l2.\ \ x \in set\ (l1@l2) = (x \in set\ l1) \vee (x \in set\ l2)$

lemma *suc-minus-swap*:
$[\![\ a <= b\ ]\!] \Longrightarrow (Suc\ b) - a = Suc\ (b - a)$

lemma *rel-pow-step*: $Re\ O\ Re\ \hat{}\ n = Re\ \hat{}\ (Suc\ n)$

lemma *min-elim*:
$(a{::}'a{::}order) < b \Longrightarrow (min\ a\ b) = a$

lemma *suc-eq-plus1*:
$Suc\ x = x + 1$

lemma *less-diff-conv2*: $((a{::}nat) + b < c) = (a < c - b)$

lemma *conjI'*:
$[\![\ X;\ X \Longrightarrow Y\ ]\!] \Longrightarrow (X \wedge Y)$

lemma *disjE2*: $a \mid b \Longrightarrow {\sim}\ a \Longrightarrow b$

lemma *set-concat-map*:
$set\ (concat\ (map\ f\ xs)) =\ \ Union\ (set\ `\ (f\ `\ set\ xs))$

lemma *doubleAllI*:
$(\forall\ x.\ P\ x = Q\ x) \Longrightarrow (\forall\ x.\ P\ x) = (\forall\ y.\ Q\ y)$

constdefs
$delL :: 'a \Rightarrow 'a\ list \Rightarrow 'a\ list$
$delL\ a\ xs == [x{\in}xs.\ x \neq a]$

lemma *delL-length*:
$\bigwedge a.\ a \in set\ xs \Longrightarrow length\ (delL\ a\ xs) < length\ xs$

lemma *delL-setdiff*: $set\ (delL\ x\ xs) = (set\ xs) - \{x\}$

lemma *setdiff-notin*:
$x \notin B \Longrightarrow (x \notin (A - B)) = (x \notin A)$

lemma *delL-subset*:
$\bigwedge x.\ set\ (delL\ x\ xs) \subseteq set\ xs$

lemma *eqSOME*: $\forall\ x.\ P\ x = Q\ x \Longrightarrow ((SOME\ x.\ P\ x) = (SOME\ y.\ Q\ y))$

lemma *not-Some-eq-pairs*:
$\forall\ a\ b.\ x \neq Some\ (a,b) \Longrightarrow x = None$

lemma *tl-drop*:
$tl\ L = drop\ (Suc\ 0)\ L$

lemma *remdups'-termination*:
$\forall\ x\ xs.\ length\ [x'{\in}xs\ .\ x' \neq x] < Suc\ (length\ xs)$
consts $remdups' :: 'a\ list \Rightarrow 'a\ list$

recdef *remdups'* *measure* $(\lambda\ xs.\ length\ xs)$
$remdups'\ [] = []$
$remdups'\ (x\#xs) = x\#(remdups'\ [x'{\in}xs.\ x' \neq x])$

(**hints** *simp*: *remdups'-termination*)

lemma *set-remdups'*:
$\bigwedge xs.\ set\ (remdups'\ xs) = set\ xs$

lemma *distinct-remdups'*:
$\bigwedge xs.\ distinct\ (remdups'\ xs)$

lemma *filter-eq*:
$\forall\ x \in set\ xs.\ P\ x = Q\ x \Longrightarrow [x{\in}\ xs.\ P\ x] = [x \in xs.\ Q\ x]$

lemma *remdups'-append*:

$\bigwedge$ *as bs. remdups' (as @ bs) = remdups' as @ [b∈ remdups' bs. b ∉ set as]*

lemma *remdups'-append-fst*:
a ∉ set as ⟹ remdups' (as@[a]@as') = remdups' as @ [a] @ ([a' ∈ (remdups' as'). a' ∉ set (as @ [a])])

lemma *in-remdups'-split*:
x ∈ set (remdups' xs) ⟹ ∃ as bs. (remdups' xs) = as @ (x#bs) ∧ x∉ set as ∧ x ∉ set bs

lemma *in-set-remdups'*:
x ∈ set xs ⟹ x ∈ set (remdups' xs)

lemma *in-rd-sp*:
x ∈ set xs ⟹ ∃ as bs. (remdups' xs) = as @ (x#bs) ∧ x∉ set as ∧ x ∉ set bs

lemma *remdups'-empty* [*simp*]:
(remdups' l = []) = (l = [])

lemma *length-remdups-set*:
length (remdups' xs) ≤ Suc 0 = (∀ x y. x ∈ set xs ∧ y ∈ set xs ⟶ x = y)

lemma *in-set-conv-decomp-fst'*:
(x ∈ set xs) = (∃ ys zs. (xs = ys @ x # zs) ∧ x ∉ set ys)

lemma *in-set-conv-decomp-fst*:
(x ∈ set xs) ⟹ (∃ ys zs. (xs = ys @ x # zs) ∧ x ∉ set ys)

lemma *in-set-conv-decomp-last*:
(x ∈ set xs) ⟹ (∃ ys zs. (xs = ys @ x # zs) ∧ x ∉ set zs)

lemma *map-parts*:
⋀ ys ys'. map f xs = ys@ys' ⟹ ∃ xs' xs''. xs = xs' @ xs'' ∧ ys = map f xs' ∧ ys' = map f xs''

lemma *option-neq*: *(Some x = None) = False*

lemma *option-neq2*:*(None = Some x) = False*
lemma *append-eq*:
(xs@xs' = xs@xs'') = (xs' = xs'')
lemma *list-set-pred*:
(∃ xs x xs'. ys = xs @ x # xs' ∧ P x) = (∃ x ∈ set ys. P x)

lemma *in-set-conv-decomp-2*:
⟦ x ∈ set s; y ∈ set s; x ≠ y ⟧ ⟹ ∃ s' s'' s'''. s = s' @ x # s'' @ y # s''' ∨ s = s' @ y # s'' @ x # s'''

lemma *list-prefix-eq*: *⋀as bs. ⟦ as @ x # as' = bs @ x # bs'; x ∉ set as; x ∉ set bs ⟧*

$\Longrightarrow as = bs$

lemma *distinct-list-match*:
$[\![\ as@[x]@as' = bs@[x]@bs';\ distinct\ (as@[x]@as');\ distinct\ (bs@[x]@bs')\ ]\!] \Longrightarrow as = bs \wedge as' = bs'$

lemma *renSnd*:
$x = x' \Longrightarrow \forall\, em.\ P\ em\ x = P\ em\ x'$

lemma *allE2*:
$[\![\ \forall\ x\ y.\ P\ x\ y;\ \ P\ x\ y \Longrightarrow R\ ]\!] \Longrightarrow R$

lemma *rev-take-nth*: $\bigwedge x\ y.\ [\![\ x < length\ s;\ y < x\ ]\!] \Longrightarrow s\ !\ (x - Suc\ y) = rev\ (take\ x\ s)\ !\ y$

lemma *split-if-False*: *if B then A else False* $\Longrightarrow B \wedge A$

lemma *forall-expand*: $(\forall\ x < Suc\ y.\ P\ x) = (P\ 0 \wedge (\forall\ x < y.\ P\ (Suc\ x)))$

lemma *not-Nil-case*:
$x \neq [] \Longrightarrow (case\ x\ of\ [] \Rightarrow a \mid l\#ls \Rightarrow b\ l\ ls) = b\ (hd\ x)\ (tl\ x)$

lemma *hd-arb*:
$hd\ [] = arbitrary$

lemma *prod-simp*:
$(a,b) = (a',b') = (a = a' \wedge b = b')$

lemma *length-filter-less*:
$aa\ mem\ S \Longrightarrow length\ [a \in S.\ a \neq aa] < length\ S$

lemma *insert-union-right*:
$insert\ p\ (A \cup B) = A \cup (insert\ p\ B)$

lemma *insert-set-filter*:
$(insert\ p\ \{x.\ x \in set\ S \wedge x \neq p\}) = (insert\ p\ (set\ S))$

lemma *intersect-mono*:
$set\ V \cap \{x.\ x \in set\ S \wedge x \neq p\} \subseteq\ \ set\ V \cap set\ S$

lemma *Un-real-subset-iff*:
$(X \cup Y) \subset Z = (X \subset Z \wedge Y \subset Z \wedge X \cup Y \neq Z)$

lemma *map-fst-tuple* :
$map\ fst\ (map\ (\lambda x.\ (x,f\ x))\ L) = L$

lemma *image-fst-tuple*: $fst\ `\ (\lambda p.\ (p,\ f\ p))\ `\ S = S$

lemma *list-length-split*:
$\bigwedge n.\ n < length\ L \Longrightarrow \exists\ ls\ l\ ls'.\ L = ls@l\#ls' \wedge length\ ls = n$

lemma *un-subset-drop*:
$(X \cup (Y \cup Z)) \subset W \Longrightarrow (X \cup Y) \subset W$
by *blast*

lemma *un-subset-drop'*:
$(X \cup (Y \cup (Z \cup (U \cup V)))) \subset W \Longrightarrow (X \cup (Z \cup U)) \subset W$

lemma *ListEq*:
$(a\#al = b\#bl) = (a{=}b \wedge al{=}bl)$

lemma *seteq-simp*:
$(\forall\ x.\ (x \in A) = (x \in B)) \Longrightarrow A = B$

lemma *map-simp2*:
$(\forall x.\ x \in set\ l \longrightarrow f\ x = f'\ x) \longrightarrow map\ f\ l = map\ f'\ l$

lemma *map-simp*:
$[\![\ l = l';\ (\forall\ x.\ x{\in}\ set\ l \longrightarrow (f\ x) = (f'\ x))\ ]\!] \Longrightarrow (map\ f\ l) = (map\ f'\ l')$

lemma *notin-simp*:
$[\![\ pc'{\notin}\ set\ l\ ]\!] \Longrightarrow pc'{\notin}\ set\ [x{\in}l.\ x{\neq}\ pc]$

lemma *neg-notin-simp*:
$[\![\ pc{\neq}pc';\ \neg\ pc'{\notin}\ set\ l\ ]\!] \Longrightarrow \neg\ pc'{\notin}\ set\ [x{\in}l.\ x{\neq}\ pc]$

lemma *option-case-simp*:
$[\![\ x \neq None\ ]\!] \Longrightarrow \exists\ a.\ x = Some\ a$

lemma *filter-list-simp*:
$[\![\ pc{\notin}\ set\ l\ ]\!] \Longrightarrow [x{\in}l.\ x{\neq}pc \wedge x{\neq}pc'] = [a{\in}l.\ a{\neq}pc']$

lemma *list-abr-simp*:
$[x{\in}S.\ x{\neq}pc \wedge x{\neq}pc'] = [x{\in}[a{\in}S.\ a \neq pc].\ x{\neq}pc']$
lemma *if-then-else-False*:
$(if\ (A{::}bool)\ then\ (B{::}bool)\ else\ False) = (A \wedge B)$
lemma *map-of-append'*:
$[\![\ x \notin set\ (map\ fst\ xs)\ ]\!] \Longrightarrow (map\text{-}of\ (xs\ @\ ys))\ x = (map\text{-}of\ ys)\ x$
lemma *triple-simp*:
$(fst\ x, fst\ (snd\ x), snd\ (snd\ x)) = x$

end