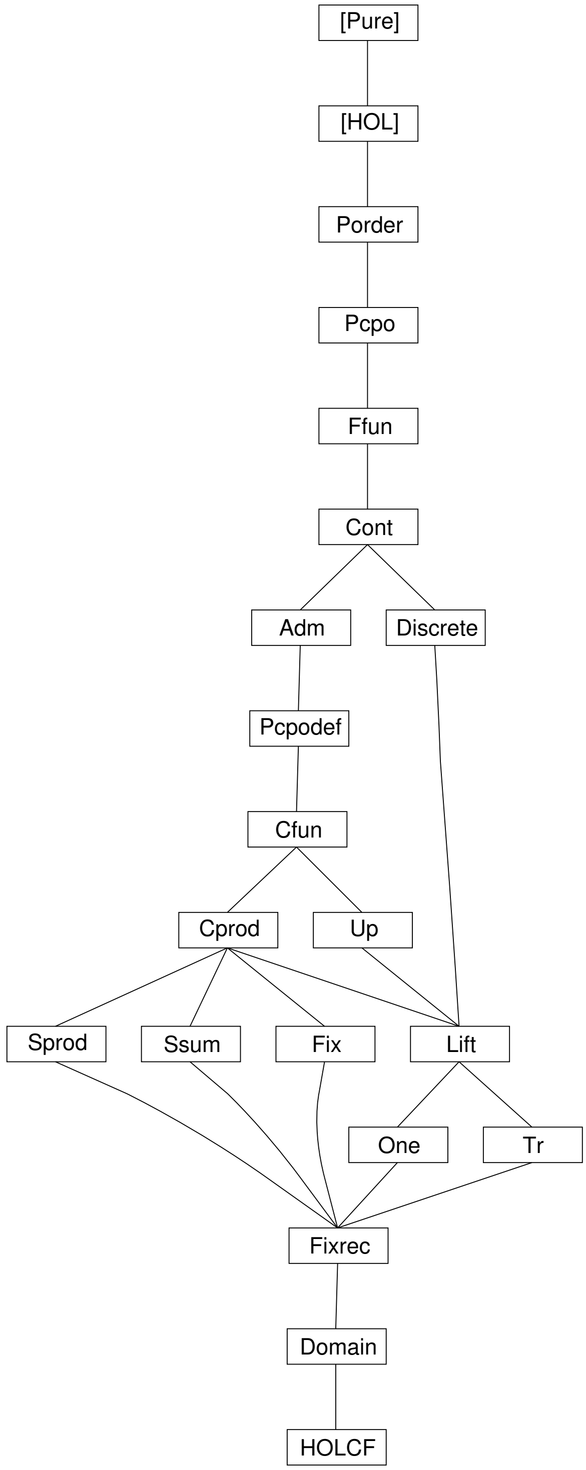
Isabelle/HOLCF — Higher-Order Logic of Computable Functions

October 1, 2005

Contents


[Pure]
[HOL]
Porder
Pcpo
Ffun
Cont
Adm
Discrete
Pcpodef
Cfun
Cprod
Up
Sprod
Ssum
Fix
Lift
One
Tr
Fixrec
Domain
HOLCF

1 Porder: Partial orders

theory *Porder*
imports *Main*
begin

1.1 Type class for partial orders

— introduce a (syntactic) class for the constant <<
axclass *sq-ord* < *type*

— characteristic constant << for po
consts
<< :: ['a,'a::sq-ord] => bool (**infixl** 55)

syntax (*xsymbols*)
op << :: ['a,'a::sq-ord] => bool (**infixl** $\sqsubseteq$ 55)

axclass *po* < *sq-ord*
— class axioms:
refl-less [*iff*]: *x* << *x*
antisym-less: [|*x* << *y*; *y* << *x* |] ==> *x* = *y*
trans-less: [|*x* << *y*; *y* << *z* |] ==> *x* << *z*

minimal fixes least element

lemma *minimal2UU*[*OF allI*] : !*x*::'*a*::*po*. *uu*<<*x* ==> *uu*=(*THE u*.!*y*. *u*<<*y*)
⟨*proof*⟩

the reverse law of anti-symmetry of *op* $\sqsubseteq$

lemma *antisym-less-inverse*: (*x*::'*a*::*po*)=*y* ==> *x* << *y* & *y* << *x*
⟨*proof*⟩

lemma *box-less*: [| (*a*::'*a*::*po*) << *b*; *c* << *a*; *b* << *d*|] ==> *c* << *d*
⟨*proof*⟩

lemma *po-eq-conv*: ((*x*::'*a*::*po*)=*y*) = (*x* << *y* & *y* << *x*)
⟨*proof*⟩

1.2 Chains and least upper bounds

consts
<| :: ['a set,'a::po] => bool (**infixl** 55)
<<| :: ['a set,'a::po] => bool (**infixl** 55)
lub :: 'a set => 'a::po
tord :: 'a::po set => bool
chain :: (nat=>'a::po) => bool
max-in-chain :: [nat,nat=>'a::po]=>bool
finite-chain :: (nat=>'a::po)=>bool

syntax
@*LUB* :: ('*b* => '*a*) => '*a* (**binder** *LUB* *10*)

translations
LUB x. t == *lub*(*range*(%*x. t*))

syntax (*xsymbols*)
LUB :: [*idts*, '*a*] => '*a* ((*3*⨆-./ -)[*0*,*10*] *10*)

defs

— class definitions
is-ub-def: *S* <| *x* == ! *y*. *y*:*S* --> *y*<<*x*
is-lub-def: *S* <<| *x* == *S* <| *x* & (!*u*. *S* <| *u* --> *x* << *u*)

— Arbitrary chains are total orders
tord-def: *tord S* == !*x y*. *x*:*S* & *y*:*S* --> (*x*<<*y* | *y*<<*x*)

— Here we use countable chains and I prefer to code them as functions!
chain-def: *chain F* == !*i*. *F i* << *F* (*Suc i*)

— finite chains, needed for monotony of continouous functions
max-in-chain-def: *max-in-chain i C* == ! *j*. *i* <= *j* --> *C*(*i*) = *C*(*j*)
finite-chain-def: *finite-chain C* == *chain*(*C*) & (? *i*. *max-in-chain i C*)

lub-def: *lub S* == (*THE x*. *S* <<| *x*)

lubs are unique

lemma *unique-lub*:
[| *S* <<| *x* ; *S* <<| *y* |] ==> *x*=*y*
⟨*proof*⟩

chains are monotone functions

lemma *chain-mono* [*rule-format*]: *chain F* ==> *x*<*y* --> *F x*<<*F y*
⟨*proof*⟩

lemma *chain-mono3*: [| *chain F*; *x* <= *y* |] ==> *F x* << *F y*
⟨*proof*⟩

The range of a chain is a totally ordered

lemma *chain-tord*: *chain*(*F*) ==> *tord*(*range*(*F*))
⟨*proof*⟩

technical lemmas about *lub* and *is-lub*

lemmas *lub* = *lub-def* [*THEN meta-eq-to-obj-eq*, *standard*]

lemma *lubI*[*OF exI*]: *EX x*. *M* <<| *x* ==> *M* <<| *lub*(*M*)
⟨*proof*⟩

lemma *thelubI*: $M <<| l ==> lub(M) = l$
⟨*proof*⟩

lemma *lub-singleton* [*simp*]: $lub\{x\} = x$
⟨*proof*⟩

access to some definition as inference rule

lemma *is-lubD1*: $S <<| x ==> S <| x$
⟨*proof*⟩

lemma *is-lub-lub*: $[| S <<| x; S <| u |] ==> x << u$
⟨*proof*⟩

lemma *is-lubI*:
$[| S <| x; !!u.\ S <| u ==> x << u |] ==> S <<| x$
⟨*proof*⟩

lemma *chainE*: $\mathit{chain}\ F ==> F(i) << F(\mathit{Suc}(i))$
⟨*proof*⟩

lemma *chainI*: $(!!i.\ F\ i << F(\mathit{Suc}\ i)) ==> \mathit{chain}\ F$
⟨*proof*⟩

lemma *chain-shift*: $\mathit{chain}\ Y ==> \mathit{chain}\ (\%i.\ Y\ (i + j))$
⟨*proof*⟩

technical lemmas about (least) upper bounds of chains

lemma *ub-rangeD*: $\mathit{range}\ S <| x\ ==> S(i) << x$
⟨*proof*⟩

lemma *ub-rangeI*: $(!!i.\ S\ i << x) ==> \mathit{range}\ S <| x$
⟨*proof*⟩

lemmas *is-ub-lub* = *is-lubD1* [*THEN ub-rangeD*, *standard*]
— $\mathit{range}\ ?S <<|\ ?x \Longrightarrow\ ?S\ ?i \sqsubseteq\ ?x$

lemma *is-ub-range-shift*:
$\mathit{chain}\ S \Longrightarrow \mathit{range}\ (\lambda i.\ S\ (i + j)) <| x = \mathit{range}\ S <| x$
⟨*proof*⟩

lemma *is-lub-range-shift*:
$\mathit{chain}\ S \Longrightarrow \mathit{range}\ (\lambda i.\ S\ (i + j)) <<| x = \mathit{range}\ S <<| x$
⟨*proof*⟩

results about finite chains

lemma *lub-finch1*:
$[| \mathit{chain}\ C; \mathit{max\text{-}in\text{-}chain}\ i\ C|] ==> \mathit{range}\ C <<| C\ i$
⟨*proof*⟩

lemma *lub-finch2*:
  $finite\text{-}chain(C)$ ==> $range(C)$ <<| $C(LEAST\ i.\ max\text{-}in\text{-}chain\ i\ C)$
⟨*proof*⟩

lemma *bin-chain*: x<<y ==> *chain* (%i. *if* i=*0* *then* x *else* y)
⟨*proof*⟩

lemma *bin-chainmax*:
  x<<y ==> *max-in-chain* (*Suc 0*) (%i. *if* (i=*0*) *then* x *else* y)
⟨*proof*⟩

lemma *lub-bin-chain*: x << y ==> *range*(%i::*nat*. *if* (i=*0*) *then* x *else* y) <<| y
⟨*proof*⟩

the maximal element in a chain is its lub

lemma *lub-chain-maxelem*: [| $Y\ i = c$; *ALL* i. $Y\ i$<<c |] ==> *lub*(*range* Y) = c
⟨*proof*⟩

the lub of a constant chain is the constant

lemma *chain-const*: *chain* ($\lambda i.\ c$)
⟨*proof*⟩

lemma *lub-const*: *range*(%x. c) <<| c
⟨*proof*⟩

lemmas *thelub-const* = *lub-const* [*THEN thelubI*, *standard*]

end

2 Pcpo: Classes cpo and pcpo

theory *Pcpo*
imports *Porder*
begin

2.1 Complete partial orders

The class cpo of chain complete partial orders

axclass *cpo* < *po*
  — class axiom:
 cpo: $chain\ S \Longrightarrow \exists x.\ range\ S$ <<| x

in cpo's everthing equal to THE lub has lub properties for every chain

lemma *thelubE*: $[\![chain\ S;\ (\bigsqcup i.\ S\ i) = (l::'a::cpo)]\!] \Longrightarrow range\ S$ <<| l
⟨*proof*⟩

Properties of the lub

lemma *is-ub-thelub*: $chain\ (S::nat \Rightarrow {'}a::cpo) \Longrightarrow S\ x \sqsubseteq (\bigsqcup i.\ S\ i)$
⟨*proof*⟩

lemma *is-lub-thelub*:
$[\![chain\ (S::nat \Rightarrow {'}a::cpo);\ range\ S <|\ x]\!] \Longrightarrow (\bigsqcup i.\ S\ i) \sqsubseteq x$
⟨*proof*⟩

lemma *lub-range-mono*:
$[\![range\ X \subseteq range\ Y;\ chain\ Y;\ chain\ (X::nat \Rightarrow {'}a::cpo)]\!]$
$\Longrightarrow (\bigsqcup i.\ X\ i) \sqsubseteq (\bigsqcup i.\ Y\ i)$
⟨*proof*⟩

lemma *lub-range-shift*:
$chain\ (Y::nat \Rightarrow {'}a::cpo) \Longrightarrow (\bigsqcup i.\ Y\ (i + j)) = (\bigsqcup i.\ Y\ i)$
⟨*proof*⟩

lemma *maxinch-is-thelub*:
$chain\ Y \Longrightarrow max\text{-}in\text{-}chain\ i\ Y = ((\bigsqcup i.\ Y\ i) = ((Y\ i)::{'}a::cpo))$
⟨*proof*⟩

the $\sqsubseteq$ relation between two chains is preserved by their lubs

lemma *lub-mono*:
$[\![chain\ (X::nat \Rightarrow {'}a::cpo);\ chain\ Y;\ \forall k.\ X\ k \sqsubseteq Y\ k]\!]$
$\Longrightarrow (\bigsqcup i.\ X\ i) \sqsubseteq (\bigsqcup i.\ Y\ i)$
⟨*proof*⟩

the = relation between two chains is preserved by their lubs

lemma *lub-equal*:
$[\![chain\ (X::nat \Rightarrow {'}a::cpo);\ chain\ Y;\ \forall k.\ X\ k = Y\ k]\!]$
$\Longrightarrow (\bigsqcup i.\ X\ i) = (\bigsqcup i.\ Y\ i)$
⟨*proof*⟩

more results about mono and = of lubs of chains

lemma *lub-mono2*:
$[\![\exists j::nat.\ \forall i>j.\ X\ i = Y\ i;\ chain\ (X::nat{=}{>}{'}a::cpo);\ chain\ Y]\!]$
$\Longrightarrow (\bigsqcup i.\ X\ i) \sqsubseteq (\bigsqcup i.\ Y\ i)$
⟨*proof*⟩

lemma *lub-equal2*:
$[\![\exists j.\ \forall i>j.\ X\ i = Y\ i;\ chain\ (X::nat \Rightarrow {'}a::cpo);\ chain\ Y]\!]$
$\Longrightarrow (\bigsqcup i.\ X\ i) = (\bigsqcup i.\ Y\ i)$
⟨*proof*⟩

lemma *lub-mono3*:
$[\![chain\ (Y::nat \Rightarrow {'}a::cpo);\ chain\ X;\ \forall i.\ \exists j.\ Y\ i \sqsubseteq X\ j]\!]$
$\Longrightarrow (\bigsqcup i.\ Y\ i) \sqsubseteq (\bigsqcup i.\ X\ i)$
⟨*proof*⟩

lemma *ch2ch-lub*:
  fixes $Y :: nat \Rightarrow nat \Rightarrow {}'a::cpo$
  assumes *1*: $\bigwedge j.\ chain\ (\lambda i.\ Y\ i\ j)$
  assumes *2*: $\bigwedge i.\ chain\ (\lambda j.\ Y\ i\ j)$
  shows $chain\ (\lambda i.\ \bigsqcup j.\ Y\ i\ j)$
⟨*proof*⟩

lemma *diag-lub*:
  fixes $Y :: nat \Rightarrow nat \Rightarrow {}'a::cpo$
  assumes *1*: $\bigwedge j.\ chain\ (\lambda i.\ Y\ i\ j)$
  assumes *2*: $\bigwedge i.\ chain\ (\lambda j.\ Y\ i\ j)$
  shows $(\bigsqcup i.\ \bigsqcup j.\ Y\ i\ j) = (\bigsqcup i.\ Y\ i\ i)$
⟨*proof*⟩

lemma *ex-lub*:
  fixes $Y :: nat \Rightarrow nat \Rightarrow {}'a::cpo$
  assumes *1*: $\bigwedge j.\ chain\ (\lambda i.\ Y\ i\ j)$
  assumes *2*: $\bigwedge i.\ chain\ (\lambda j.\ Y\ i\ j)$
  shows $(\bigsqcup i.\ \bigsqcup j.\ Y\ i\ j) = (\bigsqcup j.\ \bigsqcup i.\ Y\ i\ j)$
⟨*proof*⟩

2.2 Pointed cpos

The class pcpo of pointed cpos

axclass *pcpo* < *cpo*
  least: $\exists x.\ \forall y.\ x \sqsubseteq y$

constdefs
  $UU :: {}'a::pcpo$
  $UU \equiv THE\ x.\ ALL\ y.\ x \sqsubseteq y$

syntax (*xsymbols*)
  $UU :: {}'a::pcpo\ (\bot)$

derive the old rule minimal

lemma *UU-least*: $\forall z.\ \bot \sqsubseteq z$
⟨*proof*⟩

lemma *minimal* [*iff*]: $\bot \sqsubseteq x$
⟨*proof*⟩

lemma *UU-reorient*: $(\bot = x) = (x = \bot)$
⟨*proof*⟩

⟨*ML*⟩

useful lemmas about $\bot$

lemma *eq-UU-iff*: $(x = \bot) = (x \sqsubseteq \bot)$

⟨*proof*⟩

lemma *UU-I*: $x \sqsubseteq \bot \Longrightarrow x = \bot$
⟨*proof*⟩

lemma *not-less2not-eq*: $\neg (x::'a::po) \sqsubseteq y \Longrightarrow x \neq y$
⟨*proof*⟩

lemma *chain-UU-I*: $[\![chain\ Y;\ (\bigsqcup i.\ Y\ i) = \bot]\!] \Longrightarrow \forall i.\ Y\ i = \bot$
⟨*proof*⟩

lemma *chain-UU-I-inverse*: $\forall i::nat.\ Y\ i = \bot \Longrightarrow (\bigsqcup i.\ Y\ i) = \bot$
⟨*proof*⟩

lemma *chain-UU-I-inverse2*: $(\bigsqcup i.\ Y\ i) \neq \bot \Longrightarrow \exists i::nat.\ Y\ i \neq \bot$
⟨*proof*⟩

lemma *notUU-I*: $[\![x \sqsubseteq y;\ x \neq \bot]\!] \Longrightarrow y \neq \bot$
⟨*proof*⟩

lemma *chain-mono2*: $[\![\exists j.\ Y\ j \neq \bot;\ chain\ Y]\!] \Longrightarrow \exists j.\ \forall i{>}j.\ Y\ i \neq \bot$
⟨*proof*⟩

2.3 Chain-finite and flat cpos

further useful classes for HOLCF domains

axclass *chfin* < *po*
  chfin: $\forall Y.\ chain\ Y \longrightarrow (\exists n.\ max\text{-}in\text{-}chain\ n\ Y)$

axclass *flat* < *pcpo*
  ax-flat: $\forall x\ y.\ x \sqsubseteq y \longrightarrow (x = \bot) \vee (x = y)$

some properties for chfin and flat

chfin types are cpo

lemma *chfin-imp-cpo*:
  $chain\ (S::nat \Rightarrow 'a::chfin) \Longrightarrow \exists x.\ range\ S <<|\ x$
⟨*proof*⟩

instance *chfin* < *cpo*
⟨*proof*⟩

flat types are chfin

lemma *flat-imp-chfin*:
  $\forall Y::nat \Rightarrow 'a::flat.\ chain\ Y \longrightarrow (\exists n.\ max\text{-}in\text{-}chain\ n\ Y)$
⟨*proof*⟩

instance *flat* < *chfin*

$\langle proof \rangle$

flat subclass of chfin; *adm-flat* not needed

lemma *flat-eq*: $(a::'a::flat) \neq \bot \Longrightarrow a \sqsubseteq b = (a = b)$
$\langle proof \rangle$

lemma *chfin2finch*: $chain\ (Y::nat \Rightarrow 'a::chfin) \Longrightarrow finite\text{-}chain\ Y$
$\langle proof \rangle$

lemmata for improved admissibility introdution rule

lemma *infinite-chain-adm-lemma*:
$[\![chain\ Y;\ \forall\, i.\ P\ (Y\ i);$
$\bigwedge Y.\ [\![chain\ Y;\ \forall\, i.\ P\ (Y\ i);\ \neg\ finite\text{-}chain\ Y]\!] \Longrightarrow P\ (\bigsqcup i.\ Y\ i)]\!]$
$\Longrightarrow P\ (\bigsqcup i.\ Y\ i)$
$\langle proof \rangle$

lemma *increasing-chain-adm-lemma*:
$[\![chain\ Y;\ \ \forall\, i.\ P\ (Y\ i);\ \bigwedge Y.\ [\![chain\ Y;\ \forall\, i.\ P\ (Y\ i);$
$\forall\, i.\ \exists\, j{>}i.\ Y\ i \neq Y\ j \wedge Y\ i \sqsubseteq Y\ j]\!] \Longrightarrow P\ (\bigsqcup i.\ Y\ i)]\!]$
$\Longrightarrow P\ (\bigsqcup i.\ Y\ i)$
$\langle proof \rangle$

end

3 Ffun: Class instances for the full function space

theory *Ffun*
imports *Pcpo*
begin

3.1 Type $'a \Rightarrow 'b$ is a partial order

instance *fun* :: (*type*, *sq-ord*) *sq-ord* $\langle proof \rangle$

defs (overloaded)
less-fun-def: $(op \sqsubseteq) \equiv (\lambda f\ g.\ \forall\, x.\ f\ x \sqsubseteq g\ x)$

lemma *refl-less-fun*: $(f::'a::type \Rightarrow 'b::po) \sqsubseteq f$
$\langle proof \rangle$

lemma *antisym-less-fun*:
$[\![(f1::'a::type \Rightarrow 'b::po) \sqsubseteq f2;\ f2 \sqsubseteq f1]\!] \Longrightarrow f1 = f2$
$\langle proof \rangle$

lemma *trans-less-fun*:
$[\![(f1::'a::type \Rightarrow 'b::po) \sqsubseteq f2;\ f2 \sqsubseteq f3]\!] \Longrightarrow f1 \sqsubseteq f3$
$\langle proof \rangle$

instance *fun* :: (*type*, *po*) *po*
⟨*proof*⟩

make the symbol << accessible for type fun

lemma *less-fun*: $(f \sqsubseteq g) = (\forall x.\ f\ x \sqsubseteq g\ x)$
⟨*proof*⟩

lemma *less-fun-ext*: $(\bigwedge x.\ f\ x \sqsubseteq g\ x) \Longrightarrow f \sqsubseteq g$
⟨*proof*⟩

3.2 Type $'a \Rightarrow 'b$ is pointed

lemma *minimal-fun*: $(\lambda x.\ \bot) \sqsubseteq f$
⟨*proof*⟩

lemma *least-fun*: $\exists x{::}'a \Rightarrow 'b{::}pcpo.\ \forall y.\ x \sqsubseteq y$
⟨*proof*⟩

3.3 Type $'a \Rightarrow 'b$ is chain complete

chains of functions yield chains in the po range

lemma *ch2ch-fun*: $chain\ S \Longrightarrow chain\ (\lambda i.\ S\ i\ x)$
⟨*proof*⟩

lemma *ch2ch-fun-rev*: $(\bigwedge x.\ chain\ (\lambda i.\ S\ i\ x)) \Longrightarrow chain\ S$
⟨*proof*⟩

upper bounds of function chains yield upper bound in the po range

lemma *ub2ub-fun*:
$range\ (S{::}nat \Rightarrow 'a \Rightarrow 'b{::}po) <|\ u \Longrightarrow range\ (\lambda i.\ S\ i\ x) <|\ u\ x$
⟨*proof*⟩

Type $'a \Rightarrow 'b$ is chain complete

lemma *lub-fun*:
$chain\ (S{::}nat \Rightarrow 'a{::}type \Rightarrow 'b{::}cpo)$
$\Longrightarrow range\ S <<|\ (\lambda x.\ \bigsqcup i.\ S\ i\ x)$
⟨*proof*⟩

lemma *thelub-fun*:
$chain\ (S{::}nat \Rightarrow 'a{::}type \Rightarrow 'b{::}cpo)$
$\Longrightarrow lub\ (range\ S) = (\lambda x.\ \bigsqcup i.\ S\ i\ x)$
⟨*proof*⟩

lemma *cpo-fun*:
$chain\ (S{::}nat \Rightarrow 'a{::}type \Rightarrow 'b{::}cpo) \Longrightarrow \exists x.\ range\ S <<|\ x$
⟨*proof*⟩

instance *fun* :: (*type*, *cpo*) *cpo*
⟨*proof*⟩

instance *fun* :: (*type*, *pcpo*) *pcpo*
⟨*proof*⟩

for compatibility with old HOLCF-Version

lemma *inst-fun-pcpo*: $UU = (\%x.\ UU)$
⟨*proof*⟩

function application is strict in the left argument

lemma *app-strict* [*simp*]: $\bot\ x = \bot$
⟨*proof*⟩

end

4 Cont: Continuity and monotonicity

theory *Cont*
imports *Ffun*
begin

Now we change the default class! Form now on all untyped type variables are of default class po

defaultsort *po*

4.1 Definitions

constdefs
$monofun :: ('a \Rightarrow 'b) \Rightarrow bool$ — monotonicity
$monofun\ f \equiv \forall x\ y.\ x \sqsubseteq y \longrightarrow f\ x \sqsubseteq f\ y$

$contlub :: ('a::cpo \Rightarrow 'b::cpo) \Rightarrow bool$ — first cont. def
$contlub\ f \equiv \forall Y.\ chain\ Y \longrightarrow f\ (\bigsqcup i.\ Y\ i) = (\bigsqcup i.\ f\ (Y\ i))$

$cont :: ('a::cpo \Rightarrow 'b::cpo) \Rightarrow bool$ — secnd cont. def
$cont\ f \equiv \forall Y.\ chain\ Y \longrightarrow range\ (\lambda i.\ f\ (Y\ i)) <<|\ f\ (\bigsqcup i.\ Y\ i)$

lemma *contlubI*:
$[\![\bigwedge Y.\ chain\ Y \Longrightarrow f\ (\bigsqcup i.\ Y\ i) = (\bigsqcup i.\ f\ (Y\ i))]\!] \Longrightarrow contlub\ f$
⟨*proof*⟩

lemma *contlubE*:
$[\![contlub\ f;\ chain\ Y]\!] \Longrightarrow f\ (\bigsqcup i.\ Y\ i) = (\bigsqcup i.\ f\ (Y\ i))$
⟨*proof*⟩

lemma *contI*:
$[\![\bigwedge Y.\ chain\ Y \Longrightarrow range\ (\lambda i.\ f\ (Y\ i)) <<|\ f\ (\bigsqcup i.\ Y\ i)]\!] \Longrightarrow cont\ f$
⟨*proof*⟩

lemma *contE*:
$[\![cont\ f;\ chain\ Y]\!] \Longrightarrow range\ (\lambda i.\ f\ (Y\ i)) <<|\ f\ (\bigsqcup i.\ Y\ i)$
⟨*proof*⟩

lemma *monofunI*:
$[\![\bigwedge x\ y.\ x \sqsubseteq y \Longrightarrow f\ x \sqsubseteq f\ y]\!] \Longrightarrow monofun\ f$
⟨*proof*⟩

lemma *monofunE*:
$[\![monofun\ f;\ x \sqsubseteq y]\!] \Longrightarrow f\ x \sqsubseteq f\ y$
⟨*proof*⟩

The following results are about application for functions in $'a \Rightarrow\ 'b$

lemma *monofun-fun-fun*: $f \sqsubseteq g \Longrightarrow f\ x \sqsubseteq g\ x$
⟨*proof*⟩

lemma *monofun-fun-arg*: $[\![monofun\ f;\ x \sqsubseteq y]\!] \Longrightarrow f\ x \sqsubseteq f\ y$
⟨*proof*⟩

lemma *monofun-fun*: $[\![monofun\ f;\ monofun\ g;\ f \sqsubseteq g;\ x \sqsubseteq y]\!] \Longrightarrow f\ x \sqsubseteq g\ y$
⟨*proof*⟩

4.2 $monofun\ f \wedge contlub\ f \equiv cont\ f$

monotone functions map chains to chains

lemma *ch2ch-monofun*: $[\![monofun\ f;\ chain\ Y]\!] \Longrightarrow chain\ (\lambda i.\ f\ (Y\ i))$
⟨*proof*⟩

monotone functions map upper bound to upper bounds

lemma *ub2ub-monofun*:
$[\![monofun\ f;\ range\ Y <|\ u]\!] \Longrightarrow range\ (\lambda i.\ f\ (Y\ i)) <|\ f\ u$
⟨*proof*⟩

left to right: $monofun\ f \wedge contlub\ f \Longrightarrow cont\ f$

lemma *monocontlub2cont*: $[\![monofun\ f;\ contlub\ f]\!] \Longrightarrow cont\ f$
⟨*proof*⟩

first a lemma about binary chains

lemma *binchain-cont*:
$[\![cont\ f;\ x \sqsubseteq y]\!] \Longrightarrow range\ (\lambda i{::}nat.\ f\ (if\ i = 0\ then\ x\ else\ y)) <<|\ f\ y$
⟨*proof*⟩

right to left: $cont\ f \Longrightarrow monofun\ f \wedge contlub\ f$

part1: $cont\ f \Longrightarrow monofun\ f$

lemma *cont2mono*: $cont\ f \Longrightarrow monofun\ f$
⟨*proof*⟩

lemmas *ch2ch-cont* = *cont2mono* [*THEN ch2ch-monofun*]

right to left: $cont\ f \Longrightarrow monofun\ f \wedge contlub\ f$

part2: $cont\ f \Longrightarrow contlub\ f$

lemma *cont2contlub*: $cont\ f \Longrightarrow contlub\ f$
⟨*proof*⟩

lemmas *cont2contlubE* = *cont2contlub* [*THEN contlubE*]

4.3 Continuity of basic functions

The identity function is continuous

lemma *cont-id*: $cont\ (\lambda x.\ x)$
⟨*proof*⟩

constant functions are continuous

lemma *cont-const*: $cont\ (\lambda x.\ c)$
⟨*proof*⟩

if-then-else is continuous

lemma *cont-if*: $[\![cont\ f;\ cont\ g]\!] \Longrightarrow cont\ (\lambda x.\ if\ b\ then\ f\ x\ else\ g\ x)$
⟨*proof*⟩

4.4 Propagation of monotonicity and continuity

the lub of a chain of monotone functions is monotone

lemma *monofun-lub-fun*:
$[\![chain\ (F{::}nat \Rightarrow {'a} \Rightarrow {'b}{::}cpo);\ \forall\, i.\ monofun\ (F\ i)]\!]$
$\Longrightarrow monofun\ (\bigsqcup i.\ F\ i)$
⟨*proof*⟩

the lub of a chain of continuous functions is continuous

declare *range-composition* [*simp del*]

lemma *contlub-lub-fun*:
$[\![chain\ F;\ \forall\, i.\ cont\ (F\ i)]\!] \Longrightarrow contlub\ (\bigsqcup i.\ F\ i)$
⟨*proof*⟩

lemma *cont-lub-fun*:
$[\![chain\ F;\ \forall\, i.\ cont\ (F\ i)]\!] \Longrightarrow cont\ (\bigsqcup i.\ F\ i)$
⟨*proof*⟩

lemma *cont2cont-lub*:
$[\![chain\ F;\ \bigwedge i.\ cont\ (F\ i)]\!] \Longrightarrow cont\ (\lambda x.\ \bigsqcup i.\ F\ i\ x)$
⟨*proof*⟩

lemma *mono2mono-MF1L*: $monofun\ f \Longrightarrow monofun\ (\lambda x.\ f\ x\ y)$
⟨*proof*⟩

lemma *cont2cont-CF1L*: $cont\ f \Longrightarrow cont\ (\lambda x.\ f\ x\ y)$
⟨*proof*⟩

Note $(\lambda x.\ \lambda y.\ f\ x\ y) = f$

lemma *mono2mono-MF1L-rev*: $\forall\, y.\ monofun\ (\lambda x.\ f\ x\ y) \Longrightarrow monofun\ f$
⟨*proof*⟩

lemma *cont2cont-CF1L-rev*: $\forall\, y.\ cont\ (\lambda x.\ f\ x\ y) \Longrightarrow cont\ f$
⟨*proof*⟩

lemma *cont2cont-lambda*: $(\bigwedge y.\ cont\ (\lambda x.\ f\ x\ y)) \Longrightarrow cont\ (\lambda x.\ (\lambda y.\ f\ x\ y))$
⟨*proof*⟩

What D.A.Schmidt calls continuity of abstraction; never used here

lemma *contlub-abstraction*:
$[\![chain\ Y;\ \forall\, y.\ cont\ (\lambda x.(c::'a::cpo \Rightarrow 'b::type \Rightarrow 'c::cpo)\ x\ y)]\!] \Longrightarrow$
$(\lambda y.\ \bigsqcup i.\ c\ (Y\ i)\ y) = (\bigsqcup i.\ (\lambda y.\ c\ (Y\ i)\ y))$
⟨*proof*⟩

lemma *mono2mono-app*:
$[\![monofun\ f;\ \forall\, x.\ monofun\ (f\ x);\ monofun\ t]\!] \Longrightarrow monofun\ (\lambda x.\ (f\ x)\ (t\ x))$
⟨*proof*⟩

lemma *cont2contlub-app*:
$[\![cont\ f;\ \forall\, x.\ cont\ (f\ x);\ cont\ t]\!] \Longrightarrow contlub\ (\lambda x.\ (f\ x)\ (t\ x))$
⟨*proof*⟩

lemma *cont2cont-app*:
$[\![cont\ f;\ \forall\, x.\ cont\ (f\ x);\ cont\ t]\!] \Longrightarrow cont\ (\lambda x.\ (f\ x)\ (t\ x))$
⟨*proof*⟩

lemmas *cont2cont-app2* = *cont2cont-app* [*rule-format*]

lemma *cont2cont-app3*: $[\![cont\ f;\ cont\ t]\!] \Longrightarrow cont\ (\lambda x.\ f\ (t\ x))$
⟨*proof*⟩

4.5 Finite chains and flat pcpos

monotone functions map finite chains to finite chains

lemma *monofun-finch2finch*:
$[\![monofun\ f;\ finite\text{-}chain\ Y]\!] \Longrightarrow finite\text{-}chain\ (\lambda n.\ f\ (Y\ n))$

$\langle proof \rangle$

The same holds for continuous functions

lemma *cont-finch2finch*:
$[\![cont\ f;\ finite\text{-}chain\ Y]\!] \Longrightarrow finite\text{-}chain\ (\lambda n.\ f\ (Y\ n))$
$\langle proof \rangle$

lemma *chfindom-monofun2cont*: $monofun\ f \Longrightarrow cont\ (f::'a::chfin \Rightarrow 'b::pcpo)$
$\langle proof \rangle$

some properties of flat

lemma *flatdom-strict2mono*: $f\ \bot = \bot \Longrightarrow monofun\ (f::'a::flat \Rightarrow 'b::pcpo)$
$\langle proof \rangle$

lemma *flatdom-strict2cont*: $f\ \bot = \bot \Longrightarrow cont\ (f::'a::flat \Rightarrow 'b::pcpo)$
$\langle proof \rangle$

end

5 Adm: Admissibility

theory *Adm*
imports *Cont*
begin

defaultsort *cpo*

5.1 Definitions

constdefs
$adm :: ('a::cpo \Rightarrow bool) \Rightarrow bool$
$adm\ P \equiv \forall\, Y.\ chain\ Y \longrightarrow (\forall\, i.\ P\ (Y\ i)) \longrightarrow P\ (\bigsqcup i.\ Y\ i)$

lemma *admI*:
$(\bigwedge Y.\ [\![chain\ Y;\ \forall\, i.\ P\ (Y\ i)]\!] \Longrightarrow P\ (\bigsqcup i.\ Y\ i)) \Longrightarrow adm\ P$
$\langle proof \rangle$

lemma *triv-admI*: $\forall\, x.\ P\ x \Longrightarrow adm\ P$
$\langle proof \rangle$

lemma *admD*: $[\![adm\ P;\ chain\ Y;\ \forall\, i.\ P\ (Y\ i)]\!] \Longrightarrow P\ (\bigsqcup i.\ Y\ i)$
$\langle proof \rangle$

improved admissibility introduction

lemma *admI2*:
$(\bigwedge Y.\ [\![chain\ Y;\ \forall\, i.\ P\ (Y\ i);\ \forall\, i.\ \exists j{>}i.\ Y\ i \neq Y\ j \wedge Y\ i \sqsubseteq Y\ j]\!]$
$\Longrightarrow P\ (\bigsqcup i.\ Y\ i)) \Longrightarrow adm\ P$
$\langle proof \rangle$

5.2 Admissibility on chain-finite types

for chain-finite (easy) types every formula is admissible

lemma *adm-max-in-chain*:
$\forall\, Y.\ chain\ (Y{::}nat \Rightarrow {}'a) \longrightarrow (\exists\, n.\ max\text{-}in\text{-}chain\ n\ Y)$
$\Longrightarrow adm\ (P{::}{}'a \Rightarrow bool)$
⟨*proof*⟩

lemmas *adm-chfin* = *chfin* [*THEN adm-max-in-chain*, *standard*]

5.3 Admissibility of special formulae and propagation

lemma *adm-less*: $[\![cont\ u;\ cont\ v]\!] \Longrightarrow adm\ (\lambda x.\ u\ x \sqsubseteq v\ x)$
⟨*proof*⟩

lemma *adm-conj*: $[\![adm\ P;\ adm\ Q]\!] \Longrightarrow adm\ (\lambda x.\ P\ x \wedge Q\ x)$
⟨*proof*⟩

lemma *adm-not-free*: $adm\ (\lambda x.\ t)$
⟨*proof*⟩

lemma *adm-not-less*: $cont\ t \Longrightarrow adm\ (\lambda x.\ \neg\ t\ x \sqsubseteq u)$
⟨*proof*⟩

lemma *adm-all*: $\forall\, y.\ adm\ (P\ y) \Longrightarrow adm\ (\lambda x.\ \forall\, y.\ P\ y\ x)$
⟨*proof*⟩

lemmas *adm-all2* = *adm-all* [*rule-format*]

lemma *adm-ball*: $\forall\, y{\in}A.\ adm\ (P\ y) \Longrightarrow adm\ (\lambda x.\ \forall\, y{\in}A.\ P\ y\ x)$
⟨*proof*⟩

lemmas *adm-ball2* = *adm-ball* [*rule-format*]

lemma *adm-subst*: $[\![cont\ t;\ adm\ P]\!] \Longrightarrow adm\ (\lambda x.\ P\ (t\ x))$
⟨*proof*⟩

lemma *adm-UU-not-less*: $adm\ (\lambda x.\ \neg \perp \sqsubseteq t\ x)$
⟨*proof*⟩

lemma *adm-not-UU*: $cont\ t \Longrightarrow adm\ (\lambda x.\ \neg\ t\ x = \perp)$
⟨*proof*⟩

lemma *adm-eq*: $[\![cont\ u;\ cont\ v]\!] \Longrightarrow adm\ (\lambda x.\ u\ x = v\ x)$
⟨*proof*⟩

admissibility for disjunction is hard to prove. It takes 7 Lemmas

lemma *adm-disj-lemma1*:
$\forall\, n{::}nat.\ P\ n \vee Q\ n \Longrightarrow (\forall\, i.\ \exists\, j{\geq}i.\ P\ j) \vee (\forall\, i.\ \exists\, j{\geq}i.\ Q\ j)$

⟨*proof*⟩

lemma *adm-disj-lemma2*:
$[\![adm\ P;\ \exists X.\ chain\ X \wedge (\forall n.\ P\ (X\ n)) \wedge (\bigsqcup i.\ Y\ i) = (\bigsqcup i.\ X\ i)]\!]$
$\Longrightarrow P\ (\bigsqcup i.\ Y\ i)$
⟨*proof*⟩

lemma *adm-disj-lemma3*:
$[\![chain\ (Y{::}nat \Rightarrow 'a{::}cpo);\ \forall i.\ \exists j{\geq}i.\ P\ (Y\ j)]\!]$
$\Longrightarrow chain\ (\lambda m.\ Y\ (LEAST\ j.\ m \leq j \wedge P\ (Y\ j)))$
⟨*proof*⟩

lemma *adm-disj-lemma4*:
$[\![\forall i.\ \exists j{\geq}i.\ P\ (Y\ j)]\!] \Longrightarrow \forall m.\ P\ (Y\ (LEAST\ j{::}nat.\ m \leq j \wedge P\ (Y\ j)))$
⟨*proof*⟩

lemma *adm-disj-lemma5*:
$[\![chain\ (Y{::}nat \Rightarrow 'a{::}cpo);\ \forall i.\ \exists j{\geq}i.\ P\ (Y\ j)]\!] \Longrightarrow$
$(\bigsqcup m.\ Y\ m) = (\bigsqcup m.\ Y\ (LEAST\ j.\ m \leq j \wedge P\ (Y\ j)))$
⟨*proof*⟩

lemma *adm-disj-lemma6*:
$[\![chain\ (Y{::}nat \Rightarrow 'a{::}cpo);\ \forall i.\ \exists j{\geq}i.\ P(Y\ j)]\!] \Longrightarrow$
$\exists X.\ chain\ X \wedge (\forall n.\ P\ (X\ n)) \wedge (\bigsqcup i.\ Y\ i) = (\bigsqcup i.\ X\ i)$
⟨*proof*⟩

lemma *adm-disj-lemma7*:
$[\![adm\ P;\ chain\ Y;\ \forall i.\ \exists j{\geq}i.\ P\ (Y\ j)]\!] \Longrightarrow P\ (\bigsqcup i.\ Y\ i)$
⟨*proof*⟩

lemma *adm-disj*: $[\![adm\ P;\ adm\ Q]\!] \Longrightarrow adm\ (\lambda x.\ P\ x \vee Q\ x)$
⟨*proof*⟩

lemma *adm-imp*: $[\![adm\ (\lambda x.\ \neg\ P\ x);\ adm\ Q]\!] \Longrightarrow adm\ (\lambda x.\ P\ x \longrightarrow Q\ x)$
⟨*proof*⟩

lemma *adm-iff*:
$[\![adm\ (\lambda x.\ P\ x \longrightarrow Q\ x);\ adm\ (\lambda x.\ Q\ x \longrightarrow P\ x)]\!]$
$\Longrightarrow adm\ (\lambda x.\ P\ x = Q\ x)$
⟨*proof*⟩

lemma *adm-not-conj*:
$[\![adm\ (\lambda x.\ \neg\ P\ x);\ adm\ (\lambda x.\ \neg\ Q\ x)]\!] \Longrightarrow adm\ (\lambda x.\ \neg\ (P\ x \wedge Q\ x))$
⟨*proof*⟩

lemmas *adm-lemmas* =
adm-less adm-conj adm-not-free adm-imp adm-disj adm-eq adm-not-UU
adm-UU-not-less adm-all2 adm-not-less adm-not-conj adm-iff

declare *adm-lemmas* [*simp*]

⟨*ML*⟩

end

6 Pcpodef: Subtypes of pcpos

theory *Pcpodef*
imports *Adm*
uses (*pcpodef-package.ML*)
begin

6.1 Proving a subtype is a partial order

A subtype of a partial order is itself a partial order, if the ordering is defined in the standard way.

theorem *typedef-po*:
  fixes $Abs :: 'a::po \Rightarrow 'b::sq\text{-}ord$
  assumes *type*: *type-definition Rep Abs A*
    and *less*: $op \sqsubseteq\ \equiv \lambda x\ y.\ Rep\ x \sqsubseteq Rep\ y$
  shows $OFCLASS('b,\ po\text{-}class)$
⟨*proof*⟩

6.2 Proving a subtype is complete

A subtype of a cpo is itself a cpo if the ordering is defined in the standard way, and the defining subset is closed with respect to limits of chains. A set is closed if and only if membership in the set is an admissible predicate.

lemma *monofun-Rep*:
  assumes *less*: $op \sqsubseteq\ \equiv \lambda x\ y.\ Rep\ x \sqsubseteq Rep\ y$
  shows *monofun Rep*
⟨*proof*⟩

lemmas *ch2ch-Rep* = *ch2ch-monofun* [*OF monofun-Rep*]
lemmas *ub2ub-Rep* = *ub2ub-monofun* [*OF monofun-Rep*]

lemma *Abs-inverse-lub-Rep*:
  fixes $Abs :: 'a::cpo \Rightarrow 'b::po$
  assumes *type*: *type-definition Rep Abs A*
    and *less*: $op \sqsubseteq\ \equiv \lambda x\ y.\ Rep\ x \sqsubseteq Rep\ y$
    and *adm*: $adm\ (\lambda x.\ x \in A)$
  shows $chain\ S \Longrightarrow Rep\ (Abs\ (\bigsqcup i.\ Rep\ (S\ i))) = (\bigsqcup i.\ Rep\ (S\ i))$
⟨*proof*⟩

theorem *typedef-lub*:
  fixes $Abs :: 'a::cpo \Rightarrow 'b::po$
  assumes *type*: *type-definition Rep Abs A*
    and *less*: $op \sqsubseteq \equiv \lambda x\ y.\ Rep\ x \sqsubseteq Rep\ y$
    and *adm*: $adm\ (\lambda x.\ x \in A)$
  shows $chain\ S \Longrightarrow range\ S <<|\ Abs\ (\bigsqcup i.\ Rep\ (S\ i))$
  ⟨*proof*⟩

lemmas *typedef-thelub* = *typedef-lub* [*THEN thelubI*, *standard*]

theorem *typedef-cpo*:
  fixes $Abs :: 'a::cpo \Rightarrow 'b::po$
  assumes *type*: *type-definition Rep Abs A*
    and *less*: $op \sqsubseteq \equiv \lambda x\ y.\ Rep\ x \sqsubseteq Rep\ y$
    and *adm*: $adm\ (\lambda x.\ x \in A)$
  shows $OFCLASS('b,\ cpo\text{-}class)$
⟨*proof*⟩

6.2.1 Continuity of *Rep* and *Abs*

For any sub-cpo, the *Rep* function is continuous.

theorem *typedef-cont-Rep*:
  fixes $Abs :: 'a::cpo \Rightarrow 'b::cpo$
  assumes *type*: *type-definition Rep Abs A*
    and *less*: $op \sqsubseteq \equiv \lambda x\ y.\ Rep\ x \sqsubseteq Rep\ y$
    and *adm*: $adm\ (\lambda x.\ x \in A)$
  shows *cont Rep*
  ⟨*proof*⟩

For a sub-cpo, we can make the *Abs* function continuous only if we restrict its domain to the defining subset by composing it with another continuous function.

theorem *typedef-is-lubI*:
  assumes *less*: $op \sqsubseteq \equiv \lambda x\ y.\ Rep\ x \sqsubseteq Rep\ y$
  shows $range\ (\lambda i.\ Rep\ (S\ i)) <<|\ Rep\ x \Longrightarrow range\ S <<|\ x$
  ⟨*proof*⟩

theorem *typedef-cont-Abs*:
  fixes $Abs :: 'a::cpo \Rightarrow 'b::cpo$
  fixes $f :: 'c::cpo \Rightarrow 'a::cpo$
  assumes *type*: *type-definition Rep Abs A*
    and *less*: $op \sqsubseteq \equiv \lambda x\ y.\ Rep\ x \sqsubseteq Rep\ y$
    and *adm*: $adm\ (\lambda x.\ x \in A)$
    and *f-in-A*: $\bigwedge x.\ f\ x \in A$
    and *cont-f*: *cont f*
  shows $cont\ (\lambda x.\ Abs\ (f\ x))$
  ⟨*proof*⟩

6.3 Proving a subtype is pointed

A subtype of a cpo has a least element if and only if the defining subset has a least element.

theorem *typedef-pcpo-generic*:
  fixes $Abs :: 'a::cpo \Rightarrow 'b::cpo$
  assumes *type*: *type-definition Rep Abs A*
    and *less*: $op \sqsubseteq \equiv \lambda x\ y.\ Rep\ x \sqsubseteq Rep\ y$
    and *z-in-A*: $z \in A$
    and *z-least*: $\bigwedge x.\ x \in A \Longrightarrow z \sqsubseteq x$
  shows $OFCLASS('b,\ pcpo\text{-}class)$
⟨*proof*⟩

As a special case, a subtype of a pcpo has a least element if the defining subset contains $\bot$.

theorem *typedef-pcpo*:
  fixes $Abs :: 'a::pcpo \Rightarrow 'b::cpo$
  assumes *type*: *type-definition Rep Abs A*
    and *less*: $op \sqsubseteq \equiv \lambda x\ y.\ Rep\ x \sqsubseteq Rep\ y$
    and *UU-in-A*: $\bot \in A$
  shows $OFCLASS('b,\ pcpo\text{-}class)$
⟨*proof*⟩

6.3.1 Strictness of *Rep* and *Abs*

For a sub-pcpo where $\bot$ is a member of the defining subset, *Rep* and *Abs* are both strict.

theorem *typedef-Abs-strict*:
  assumes *type*: *type-definition Rep Abs A*
    and *less*: $op \sqsubseteq \equiv \lambda x\ y.\ Rep\ x \sqsubseteq Rep\ y$
    and *UU-in-A*: $\bot \in A$
  shows $Abs\ \bot = \bot$
⟨*proof*⟩

theorem *typedef-Rep-strict*:
  assumes *type*: *type-definition Rep Abs A*
    and *less*: $op \sqsubseteq \equiv \lambda x\ y.\ Rep\ x \sqsubseteq Rep\ y$
    and *UU-in-A*: $\bot \in A$
  shows $Rep\ \bot = \bot$
⟨*proof*⟩

theorem *typedef-Abs-defined*:
  assumes *type*: *type-definition Rep Abs A*
    and *less*: $op \sqsubseteq \equiv \lambda x\ y.\ Rep\ x \sqsubseteq Rep\ y$
    and *UU-in-A*: $\bot \in A$
  shows $[\![x \neq \bot;\ x \in A]\!] \Longrightarrow Abs\ x \neq \bot$
⟨*proof*⟩

theorem *typedef-Rep-defined*:
  assumes *type*: *type-definition Rep Abs A*
    and *less*: $op \sqsubseteq \equiv \lambda x\ y.\ Rep\ x \sqsubseteq Rep\ y$
    and *UU-in-A*: $\perp \in A$
  shows $x \neq \perp \Longrightarrow Rep\ x \neq \perp$
⟨*proof*⟩

6.4 HOLCF type definition package

⟨*ML*⟩

end

7 Cfun: The type of continuous functions

theory *Cfun*
imports *Pcpodef*
uses (*cont-proc.ML*)
begin

defaultsort *cpo*

7.1 Definition of continuous function type

lemma *Ex-cont*: $\exists f.\ cont\ f$
⟨*proof*⟩

lemma *adm-cont*: *adm cont*
⟨*proof*⟩

cpodef (*CFun*) ('*a*, '*b*) −> (**infixr** *0*) = {*f*::'*a* => '*b*. *cont f*}
⟨*proof*⟩

syntax
  Rep-CFun :: ('*a* −> '*b*) => ('*a* => '*b*) (-$- [*999*,*1000*] *999*)

  Abs-CFun :: ('*a* => '*b*) => ('*a* −> '*b*) (**binder** *LAM* *10*)

syntax (*xsymbols*)
  −> :: [*type*, *type*] => *type* ((- →/ -) [*1*,*0*]*0*)
  LAM :: [*idts*, '*a* => '*b*] => ('*a* −> '*b*)
    ((*3*Λ-./ -) [*0*, *10*] *10*)
  Rep-CFun :: ('*a* −> '*b*) => ('*a* => '*b*) ((-·-) [*999*,*1000*] *999*)

syntax (*HTML* **output**)
  Rep-CFun :: ('*a* −> '*b*) => ('*a* => '*b*) ((-·-) [*999*,*1000*] *999*)

7.2 Class instances

lemma *UU-CFun*: $\perp \in$ *CFun*
⟨*proof*⟩

instance $->$ :: (*cpo*, *pcpo*) *pcpo*
⟨*proof*⟩

lemmas *Rep-CFun-strict* =
typedef-Rep-strict [*OF type-definition-CFun less-CFun-def UU-CFun*]

lemmas *Abs-CFun-strict* =
typedef-Abs-strict [*OF type-definition-CFun less-CFun-def UU-CFun*]

Additional lemma about the isomorphism between $'a \to 'b$ and *CFun*

lemma *Abs-CFun-inverse2*: *cont f* $\Longrightarrow$ *Rep-CFun* (*Abs-CFun f*) = *f*
⟨*proof*⟩

Beta-equality for continuous functions

lemma *beta-cfun* [*simp*]: *cont f* $\Longrightarrow (\Lambda\ x.\ f\ x)\cdot u = f\ u$
⟨*proof*⟩

Eta-equality for continuous functions

lemma *eta-cfun*: $(\Lambda\ x.\ f\cdot x) = f$
⟨*proof*⟩

Extensionality for continuous functions

lemma *ext-cfun*: $(\bigwedge x.\ f\cdot x = g\cdot x) \Longrightarrow f = g$
⟨*proof*⟩

lemmas about application of continuous functions

lemma *cfun-cong*: $[\![f = g;\ x = y]\!] \Longrightarrow f\cdot x = g\cdot y$
⟨*proof*⟩

lemma *cfun-fun-cong*: $f = g \Longrightarrow f\cdot x = g\cdot x$
⟨*proof*⟩

lemma *cfun-arg-cong*: $x = y \Longrightarrow f\cdot x = f\cdot y$
⟨*proof*⟩

7.3 Continuity of application

lemma *cont-Rep-CFun1*: *cont* $(\lambda f.\ f\cdot x)$
⟨*proof*⟩

lemma *cont-Rep-CFun2*: *cont* $(\lambda x.\ f\cdot x)$
⟨*proof*⟩

lemmas *monofun-Rep-CFun* = *cont-Rep-CFun* [*THEN cont2mono*]

lemmas *contlub-Rep-CFun* = *cont-Rep-CFun* [*THEN cont2contlub*]

lemmas *monofun-Rep-CFun1* = *cont-Rep-CFun1* [*THEN cont2mono*, *standard*]
lemmas *contlub-Rep-CFun1* = *cont-Rep-CFun1* [*THEN cont2contlub*, *standard*]
lemmas *monofun-Rep-CFun2* = *cont-Rep-CFun2* [*THEN cont2mono*, *standard*]
lemmas *contlub-Rep-CFun2* = *cont-Rep-CFun2* [*THEN cont2contlub*, *standard*]

contlub, cont properties of *Rep-CFun* in each argument

lemma *contlub-cfun-arg*: $chain\ Y \Longrightarrow f\cdot(lub\ (range\ Y)) = (\bigsqcup i.\ f\cdot(Y\ i))$
⟨*proof*⟩

lemma *cont-cfun-arg*: $chain\ Y \Longrightarrow range\ (\lambda i.\ f\cdot(Y\ i)) <<|\ f\cdot(lub\ (range\ Y))$
⟨*proof*⟩

lemma *contlub-cfun-fun*: $chain\ F \Longrightarrow lub\ (range\ F)\cdot x = (\bigsqcup i.\ F\ i\cdot x)$
⟨*proof*⟩

lemma *cont-cfun-fun*: $chain\ F \Longrightarrow range\ (\lambda i.\ F\ i\cdot x) <<|\ lub\ (range\ F)\cdot x$
⟨*proof*⟩

Extensionality wrt. *op* $\sqsubseteq$ in $'a \to\ 'b$

lemma *less-cfun-ext*: $(\bigwedge x.\ f\cdot x \sqsubseteq g\cdot x) \Longrightarrow f \sqsubseteq g$
⟨*proof*⟩

monotonicity of application

lemma *monofun-cfun-fun*: $f \sqsubseteq g \Longrightarrow f\cdot x \sqsubseteq g\cdot x$
⟨*proof*⟩

lemma *monofun-cfun-arg*: $x \sqsubseteq y \Longrightarrow f\cdot x \sqsubseteq f\cdot y$
⟨*proof*⟩

lemma *monofun-cfun*: $[\![f \sqsubseteq g;\ x \sqsubseteq y]\!] \Longrightarrow f\cdot x \sqsubseteq g\cdot y$
⟨*proof*⟩

ch2ch - rules for the type $'a \to\ 'b$

lemma *chain-monofun*: $chain\ Y \Longrightarrow chain\ (\lambda i.\ f\cdot(Y\ i))$
⟨*proof*⟩

lemma *ch2ch-Rep-CFunR*: $chain\ Y \Longrightarrow chain\ (\lambda i.\ f\cdot(Y\ i))$
⟨*proof*⟩

lemma *ch2ch-Rep-CFunL*: $chain\ F \Longrightarrow chain\ (\lambda i.\ (F\ i)\cdot x)$
⟨*proof*⟩

lemma *ch2ch-Rep-CFun*: $[\![chain\ F;\ chain\ Y]\!] \Longrightarrow chain\ (\lambda i.\ (F\ i)\cdot(Y\ i))$
⟨*proof*⟩

contlub, cont properties of *Rep-CFun* in both arguments

lemma *contlub-cfun*:
$[\![chain\ F;\ chain\ Y]\!] \Longrightarrow (\bigsqcup i.\ F\ i)\cdot(\bigsqcup i.\ Y\ i) = (\bigsqcup i.\ F\ i\cdot(Y\ i))$
⟨*proof*⟩

lemma *cont-cfun*:
$[\![chain\ F;\ chain\ Y]\!] \Longrightarrow range\ (\lambda i.\ F\ i\cdot(Y\ i)) <<|\ (\bigsqcup i.\ F\ i)\cdot(\bigsqcup i.\ Y\ i)$
⟨*proof*⟩

strictness

lemma *strictI*: $f\cdot x = \bot \Longrightarrow f\cdot\bot = \bot$
⟨*proof*⟩

the lub of a chain of continous functions is monotone

lemma *lub-cfun-mono*: $chain\ F \Longrightarrow monofun\ (\lambda x.\ \bigsqcup i.\ F\ i\cdot x)$
⟨*proof*⟩

a lemma about the exchange of lubs for type $'a \rightarrow\ 'b$

lemma *ex-lub-cfun*:
$[\![chain\ F;\ chain\ Y]\!] \Longrightarrow (\bigsqcup j.\ \bigsqcup i.\ F\ j\cdot(Y\ i)) = (\bigsqcup i.\ \bigsqcup j.\ F\ j\cdot(Y\ i))$
⟨*proof*⟩

the lub of a chain of cont. functions is continuous

lemma *cont-lub-cfun*: $chain\ F \Longrightarrow cont\ (\lambda x.\ \bigsqcup i.\ F\ i\cdot x)$
⟨*proof*⟩

type $'a \rightarrow\ 'b$ is chain complete

lemma *lub-cfun*: $chain\ F \Longrightarrow range\ F <<|\ (\Lambda\ x.\ \bigsqcup i.\ F\ i\cdot x)$
⟨*proof*⟩

lemma *thelub-cfun*: $chain\ F \Longrightarrow lub\ (range\ F) = (\Lambda\ x.\ \bigsqcup i.\ F\ i\cdot x)$
⟨*proof*⟩

7.4 Miscellaneous

Monotonicity of *Abs-CFun*

lemma *semi-monofun-Abs-CFun*:
$[\![cont\ f;\ cont\ g;\ f \sqsubseteq g]\!] \Longrightarrow Abs\text{-}CFun\ f \sqsubseteq Abs\text{-}CFun\ g$
⟨*proof*⟩

for compatibility with old HOLCF-Version

lemma *inst-cfun-pcpo*: $\bot = (\Lambda\ x.\ \bot)$
⟨*proof*⟩

7.5 Continuity of application

cont2cont lemma for *Rep-CFun*

lemma *cont2cont-Rep-CFun*:

$\llbracket cont\ f;\ cont\ t\rrbracket \Longrightarrow cont\ (\lambda x.\ (f\ x){\cdot}(t\ x))$
⟨*proof*⟩

cont2mono Lemma for $\lambda x.\ \Lambda y.\ c1\ x\ y$

lemma *cont2mono-LAM*:
assumes *p1*: !!*x*. *cont*(*c1 x*)
assumes *p2*: !!*y*. *monofun*(%*x*. *c1 x y*)
shows *monofun*(%*x*. *LAM y*. *c1 x y*)
⟨*proof*⟩

cont2cont Lemma for $\lambda x.\ \Lambda y.\ c1\ x\ y$

lemma *cont2cont-LAM*:
assumes *p1*: !!*x*. *cont*(*c1 x*)
assumes *p2*: !!*y*. *cont*(%*x*. *c1 x y*)
shows *cont*(%*x*. *LAM y*. *c1 x y*)
⟨*proof*⟩

continuity simplification procedure

lemmas *cont-lemmas1* =
cont-const cont-id cont-Rep-CFun2 cont2cont-Rep-CFun cont2cont-LAM

⟨*ML*⟩

function application is strict in its first argument

lemma *Rep-CFun-strict1* [*simp*]: $\bot{\cdot}x = \bot$
⟨*proof*⟩

some lemmata for functions with flat/chfin domain/range types

lemma *chfin-Rep-CFunR*: *chain* (*Y*::*nat* => ′*a*::*cpo*−>′*b*::*chfin*)
==> !*s*. ? *n*. *lub*(*range*(*Y*))$*s* = *Y n*$*s*
⟨*proof*⟩

7.6 Continuous injection-retraction pairs

Continuous retractions are strict.

lemma *retraction-strict*:
$\forall x.\ f{\cdot}(g{\cdot}x) = x \Longrightarrow f{\cdot}\bot = \bot$
⟨*proof*⟩

lemma *injection-eq*:
$\forall x.\ f{\cdot}(g{\cdot}x) = x \Longrightarrow (g{\cdot}x = g{\cdot}y) = (x = y)$
⟨*proof*⟩

lemma *injection-less*:
$\forall x.\ f{\cdot}(g{\cdot}x) = x \Longrightarrow (g{\cdot}x \sqsubseteq g{\cdot}y) = (x \sqsubseteq y)$
⟨*proof*⟩

lemma *injection-defined-rev*:
$[\![\forall x.\ f\cdot(g\cdot x) = x;\ g\cdot z = \bot]\!] \Longrightarrow z = \bot$
⟨*proof*⟩

lemma *injection-defined*:
$[\![\forall x.\ f\cdot(g\cdot x) = x;\ z \neq \bot]\!] \Longrightarrow g\cdot z \neq \bot$
⟨*proof*⟩

propagation of flatness and chain-finiteness by retractions

lemma *chfin2chfin*:
$\forall y.\ (f::'a::chfin \to 'b)\cdot(g\cdot y) = y$
$\Longrightarrow \forall Y::nat \Rightarrow 'b.\ chain\ Y \longrightarrow (\exists n.\ max\text{-}in\text{-}chain\ n\ Y)$
⟨*proof*⟩

lemma *flat2flat*:
$\forall y.\ (f::'a::flat \to 'b::pcpo)\cdot(g\cdot y) = y$
$\Longrightarrow \forall x\ y::'b.\ x \sqsubseteq y \longrightarrow x = \bot \vee x = y$
⟨*proof*⟩

a result about functions with flat codomain

lemma *flat-eqI*: $[\![(x::'a::flat) \sqsubseteq y;\ x \neq \bot]\!] \Longrightarrow x = y$
⟨*proof*⟩

lemma *flat-codom*:
$f\cdot x = (c::'b::flat) \Longrightarrow f\cdot\bot = \bot \vee (\forall z.\ f\cdot z = c)$
⟨*proof*⟩

7.7 Identity and composition

consts
ID $::\ 'a \to 'a$
cfcomp $::\ ('b \to 'c) \to ('a \to 'b) \to 'a \to 'c$

syntax $@oo ::\ ['b \to 'c,\ 'a \to 'b] \Rightarrow 'a \to 'c$ (**infixr** *oo 100*)

translations *f1 oo f2* == *cfcomp$f1$f2*

defs
ID-def: $ID \equiv (\Lambda\ x.\ x)$
oo-def: $cfcomp \equiv (\Lambda\ f\ g\ x.\ f\cdot(g\cdot x))$

lemma *ID1* [*simp*]: $ID\cdot x = x$
⟨*proof*⟩

lemma *cfcomp1*: $(f\ oo\ g) = (\Lambda\ x.\ f\cdot(g\cdot x))$
⟨*proof*⟩

lemma *cfcomp2* [*simp*]: $(f\ oo\ g)\cdot x = f\cdot(g\cdot x)$
⟨*proof*⟩

Show that interpretation of (pcpo,-->-) is a category. The class of objects is interpretation of syntactical class pcpo. The class of arrows between objects $'a$ and $'b$ is interpret. of $'a \to 'b$. The identity arrow is interpretation of *ID*. The composition of f and g is interpretation of *oo*.

lemma *ID2* [*simp*]: *f oo ID* = *f*
⟨*proof*⟩

lemma *ID3* [*simp*]: *ID oo f* = *f*
⟨*proof*⟩

lemma *assoc-oo*: *f oo* (*g oo h*) = (*f oo g*) *oo h*
⟨*proof*⟩

7.8 Strictified functions

defaultsort *pcpo*

consts
Istrictify :: $('a \to 'b) \Rightarrow 'a \Rightarrow 'b$
strictify :: $('a \to 'b) \to 'a \to 'b$

defs
Istrictify-def: *Istrictify f x* ≡ *if x* = ⊥ *then* ⊥ *else f·x*
strictify-def: *strictify* ≡ (Λ *f x*. *Istrictify f x*)

results about strictify

lemma *Istrictify1*: *Istrictify f* ⊥ = ⊥
⟨*proof*⟩

lemma *Istrictify2*: *x* ≠ ⊥ ⟹ *Istrictify f x* = *f·x*
⟨*proof*⟩

lemma *cont-Istrictify1*: *cont* (λ*f*. *Istrictify f x*)
⟨*proof*⟩

lemma *monofun-Istrictify2*: *monofun* (λ*x*. *Istrictify f x*)
⟨*proof*⟩

lemma *contlub-Istrictify2*: *contlub* (λ*x*. *Istrictify f x*)
⟨*proof*⟩

lemmas *cont-Istrictify2* =
monocontlub2cont [*OF monofun-Istrictify2 contlub-Istrictify2*, *standard*]

lemma *strictify1* [*simp*]: *strictify·f·*⊥ = ⊥
⟨*proof*⟩

lemma *strictify2* [*simp*]: *x* ≠ ⊥ ⟹ *strictify·f·x* = *f·x*
⟨*proof*⟩

lemma *strictify-conv-if*: $strictify \cdot f \cdot x = (if\ x = \bot\ then\ \bot\ else\ f \cdot x)$
⟨*proof*⟩

end

8 Cprod: The cpo of cartesian products

theory *Cprod*
imports *Cfun*
begin

defaultsort *cpo*

8.1 Type *unit* is a pcpo

instance *unit* :: *sq-ord* ⟨*proof*⟩

defs (overloaded)
less-unit-def [*simp*]: $x \sqsubseteq (y::unit) \equiv True$

instance *unit* :: *po*
⟨*proof*⟩

instance *unit* :: *cpo*
⟨*proof*⟩

instance *unit* :: *pcpo*
⟨*proof*⟩

8.2 Type $'a \times 'b$ is a partial order

instance $*$ :: (*sq-ord*, *sq-ord*) *sq-ord* ⟨*proof*⟩

defs (overloaded)
less-cprod-def: $(op \sqsubseteq) \equiv \lambda p1\ p2.\ (fst\ p1 \sqsubseteq fst\ p2 \wedge snd\ p1 \sqsubseteq snd\ p2)$

lemma *refl-less-cprod*: $(p::'a * 'b) \sqsubseteq p$
⟨*proof*⟩

lemma *antisym-less-cprod*: $[\![(p1::'a * 'b) \sqsubseteq p2;\ p2 \sqsubseteq p1]\!] \Longrightarrow p1 = p2$
⟨*proof*⟩

lemma *trans-less-cprod*: $[\![(p1::'a*'b) \sqsubseteq p2;\ p2 \sqsubseteq p3]\!] \Longrightarrow p1 \sqsubseteq p3$
⟨*proof*⟩

instance $*$ :: (*cpo*, *cpo*) *po*
⟨*proof*⟩

8.3 Monotonicity of (-,-), *fst*, *snd*

Pair (-,-) is monotone in both arguments

lemma *monofun-pair1*: *monofun* $(\lambda x.\ (x,\ y))$
⟨*proof*⟩

lemma *monofun-pair2*: *monofun* $(\lambda y.\ (x,\ y))$
⟨*proof*⟩

lemma *monofun-pair*:
$[\![x1 \sqsubseteq x2;\ y1 \sqsubseteq y2]\!] \Longrightarrow (x1,\ y1) \sqsubseteq (x2,\ y2)$
⟨*proof*⟩

fst and *snd* are monotone

lemma *monofun-fst*: *monofun fst*
⟨*proof*⟩

lemma *monofun-snd*: *monofun snd*
⟨*proof*⟩

8.4 Type $'a \times 'b$ is a cpo

lemma *lub-cprod*:
$chain\ S \Longrightarrow range\ S <<|\ (\bigsqcup i.\ fst\ (S\ i),\ \bigsqcup i.\ snd\ (S\ i))$
⟨*proof*⟩

lemma *thelub-cprod*:
$chain\ S \Longrightarrow lub\ (range\ S) = (\bigsqcup i.\ fst\ (S\ i),\ \bigsqcup i.\ snd\ (S\ i))$
⟨*proof*⟩

lemma *cpo-cprod*:
$chain\ (S::nat \Rightarrow 'a::cpo * 'b::cpo) \Longrightarrow \exists x.\ range\ S <<|\ x$
⟨*proof*⟩

instance $*$ :: (*cpo*, *cpo*) *cpo*
⟨*proof*⟩

8.5 Type $'a \times 'b$ is pointed

lemma *minimal-cprod*: $(\bot,\ \bot) \sqsubseteq p$
⟨*proof*⟩

lemma *least-cprod*: $EX\ x::'a::pcpo * 'b::pcpo.\ ALL\ y.\ x \sqsubseteq y$
⟨*proof*⟩

instance $*$ :: (*pcpo*, *pcpo*) *pcpo*
⟨*proof*⟩

for compatibility with old HOLCF-Version

lemma *inst-cprod-pcpo*: $UU = (UU,UU)$
⟨*proof*⟩

8.6 Continuity of (-,-), *fst*, *snd*

lemma *contlub-pair1*: *contlub* $(\lambda x. (x, y))$
⟨*proof*⟩

lemma *contlub-pair2*: *contlub* $(\lambda y. (x, y))$
⟨*proof*⟩

lemma *cont-pair1*: *cont* $(\lambda x. (x, y))$
⟨*proof*⟩

lemma *cont-pair2*: *cont* $(\lambda y. (x, y))$
⟨*proof*⟩

lemma *contlub-fst*: *contlub fst*
⟨*proof*⟩

lemma *contlub-snd*: *contlub snd*
⟨*proof*⟩

lemma *cont-fst*: *cont fst*
⟨*proof*⟩

lemma *cont-snd*: *cont snd*
⟨*proof*⟩

8.7 Continuous versions of constants

consts
  cpair :: $'a \to 'b \to ('a * 'b)$
  cfst :: $('a * 'b) \to 'a$
  csnd :: $('a * 'b) \to 'b$
  csplit :: $('a \to 'b \to 'c) \to ('a * 'b) \to 'c$

syntax
  @*ctuple* :: $['a, args] \Rightarrow 'a * 'b$ ((*1*<-,/ ->))

translations
  <*x*, *y*, *z*> == <*x*, <*y*, *z*>>
  <*x*, *y*> == *cpair*\$*x*\$*y*

defs
  cpair-def: *cpair* $\equiv (\Lambda\ x\ y. (x, y))$
  cfst-def: *cfst* $\equiv (\Lambda\ p.\ fst\ p)$
  csnd-def: *csnd* $\equiv (\Lambda\ p.\ snd\ p)$
  csplit-def: *csplit* $\equiv (\Lambda\ f\ p.\ f\cdot(cfst\cdot p)\cdot(csnd\cdot p))$

8.8 Syntax

syntax for *LAM <x,y,z>.e*

syntax
-LAM :: [*patterns*, *'a* ⇒ *'b*] ⇒ (*'a* → *'b*) ((*3LAM* <->./ -) [*0*, *10*] *10*)

translations
LAM <x,y,zs>. b == *csplit*$(*LAM x. LAM <y,zs>. b*)
LAM <x,y>. LAM zs. b <= *csplit*$(*LAM x y zs. b*)
LAM <x,y>.b == *csplit*$(*LAM x y. b*)

syntax (*xsymbols*)
-LAM :: [*patterns*, *'a* => *'b*] => (*'a* −> *'b*) ((*3*Λ()<->./ -) [*0*, *10*] *10*)

syntax for Let

constdefs
CLet :: *'a* → (*'a* → *'b*) → *'b*
CLet ≡ Λ *s f*. *f*·*s*

nonterminals
Cletbinds Cletbind

syntax
-Cbind :: [*pttrn*, *'a*] => *Cletbind* ((*2*- =/ -) *10*)
-Cbindp :: [*patterns*, *'a*] => *Cletbind* ((*2*<-> =/ -) *10*)
:: *Cletbind* => *Cletbinds* (-)
-Cbinds :: [*Cletbind*, *Cletbinds*] => *Cletbinds* (-;/ -)
-CLet :: [*Cletbinds*, *'a*] => *'a* ((*Let* (-)/ *in* (-)) *10*)

translations
-CLet (*-Cbinds b bs*) *e* == *-CLet b* (*-CLet bs e*)
Let x = *a in LAM ys. e* == *CLet*$*a*$(*LAM x ys. e*)
Let x = *a in e* == *CLet*$*a*$(*LAM x. e*)
Let <xs> = *a in e* == *CLet*$*a*$(*LAM <xs>. e*)

8.9 Convert all lemmas to the continuous versions

lemma *cpair-eq-pair*: <*x*, *y*> = (*x*, *y*)
⟨*proof*⟩

lemma *inject-cpair*: <*a*,*b*> = <*aa*,*ba*> ⟹ *a* = *aa* ∧ *b* = *ba*
⟨*proof*⟩

lemma *cpair-eq* [*iff*]: (<*a*, *b*> = <*a'*, *b'*>) = (*a* = *a'* ∧ *b* = *b'*)
⟨*proof*⟩

lemma *cpair-less*: (<*a*, *b*> ⊑ <*a'*, *b'*>) = (*a* ⊑ *a'* ∧ *b* ⊑ *b'*)
⟨*proof*⟩

lemma *cpair-defined-iff*: $(<x,\ y> = \bot) = (x = \bot \wedge y = \bot)$
$\langle \mathit{proof} \rangle$

lemma *cpair-strict*: $<\bot,\ \bot> = \bot$
$\langle \mathit{proof} \rangle$

lemma *inst-cprod-pcpo2*: $\bot = <\bot,\ \bot>$
$\langle \mathit{proof} \rangle$

lemma *defined-cpair-rev*:
$<a,b> = \bot \Longrightarrow a = \bot \wedge b = \bot$
$\langle \mathit{proof} \rangle$

lemma *Exh-Cprod2*: $\exists\, a\ b.\ z = <a,\ b>$
$\langle \mathit{proof} \rangle$

lemma *cprodE*: $[\![\bigwedge x\ y.\ p = <x,\ y> \Longrightarrow Q]\!] \Longrightarrow Q$
$\langle \mathit{proof} \rangle$

lemma *cfst-cpair* [*simp*]: $\mathit{cfst}\cdot<x,\ y> = x$
$\langle \mathit{proof} \rangle$

lemma *csnd-cpair* [*simp*]: $\mathit{csnd}\cdot<x,\ y> = y$
$\langle \mathit{proof} \rangle$

lemma *cfst-strict* [*simp*]: $\mathit{cfst}\cdot\bot = \bot$
$\langle \mathit{proof} \rangle$

lemma *csnd-strict* [*simp*]: $\mathit{csnd}\cdot\bot = \bot$
$\langle \mathit{proof} \rangle$

lemma *surjective-pairing-Cprod2*: $<\mathit{cfst}\cdot p,\ \mathit{csnd}\cdot p> = p$
$\langle \mathit{proof} \rangle$

lemma *less-cprod*: $x \sqsubseteq y = (\mathit{cfst}\cdot x \sqsubseteq \mathit{cfst}\cdot y \wedge \mathit{csnd}\cdot x \sqsubseteq \mathit{csnd}\cdot y)$
$\langle \mathit{proof} \rangle$

lemma *eq-cprod*: $(x = y) = (\mathit{cfst}\cdot x = \mathit{cfst}\cdot y \wedge \mathit{csnd}\cdot x = \mathit{csnd}\cdot y)$
$\langle \mathit{proof} \rangle$

lemma *lub-cprod2*:
$\mathit{chain}\ S \Longrightarrow \mathit{range}\ S <<|\ <\bigsqcup i.\ \mathit{cfst}\cdot(S\ i),\ \bigsqcup i.\ \mathit{csnd}\cdot(S\ i)>$
$\langle \mathit{proof} \rangle$

lemma *thelub-cprod2*:
$\mathit{chain}\ S \Longrightarrow \mathit{lub}\ (\mathit{range}\ S) = <\bigsqcup i.\ \mathit{cfst}\cdot(S\ i),\ \bigsqcup i.\ \mathit{csnd}\cdot(S\ i)>$
$\langle \mathit{proof} \rangle$

lemma *csplit2* [*simp*]: $\mathit{csplit}\cdot f\cdot<x,y> = f\cdot x\cdot y$

⟨*proof*⟩

lemma *csplit3* [*simp*]: $csplit \cdot cpair \cdot z = z$
⟨*proof*⟩

lemmas *Cprod-rews* = *cfst-cpair csnd-cpair csplit2*

end

9 Sprod: The type of strict products

theory *Sprod*
imports *Cprod*
begin

defaultsort *pcpo*

9.1 Definition of strict product type

pcpodef (*Sprod*) $('a, 'b)$ ** (**infixr** *20*) =
$\{p{::}'a \times 'b.\ p = \bot \vee (cfst \cdot p \neq \bot \wedge csnd \cdot p \neq \bot)\}$
⟨*proof*⟩

syntax (*xsymbols*)
** :: [*type*, *type*] => *type* ((- ⊗/ -) [*21*,*20*] *20*)
syntax (*HTML* **output**)
** :: [*type*, *type*] => *type* ((- ⊗/ -) [*21*,*20*] *20*)

lemma *spair-lemma*:
$<strictify \cdot (\Lambda\ b.\ a) \cdot b,\ strictify \cdot (\Lambda\ a.\ b) \cdot a> \in Sprod$
⟨*proof*⟩

9.2 Definitions of constants

consts
$sfst :: ('a ** 'b) \to 'a$
$ssnd :: ('a ** 'b) \to 'b$
$spair :: 'a \to 'b \to ('a ** 'b)$
$ssplit :: ('a \to 'b \to 'c) \to ('a ** 'b) \to 'c$

defs
sfst-def: $sfst \equiv \Lambda\ p.\ cfst \cdot (Rep\text{-}Sprod\ p)$
ssnd-def: $ssnd \equiv \Lambda\ p.\ csnd \cdot (Rep\text{-}Sprod\ p)$
spair-def: $spair \equiv \Lambda\ a\ b.\ Abs\text{-}Sprod$
$<strictify \cdot (\Lambda\ b.\ a) \cdot b,\ strictify \cdot (\Lambda\ a.\ b) \cdot a>$
ssplit-def: $ssplit \equiv \Lambda\ f.\ strictify \cdot (\Lambda\ p.\ f \cdot (sfst \cdot p) \cdot (ssnd \cdot p))$

syntax

$@stuple$ :: $['a,\ args] => 'a ** 'b$ $((1'(:\text{-},/\ \text{-}:')))$

translations

$(:x,\ y,\ z:)$ == $(:x,\ (:y,\ z:):)$
$(:x,\ y:)$ == $spair\$x\$y$

9.3 Case analysis

lemma *spair-Abs-Sprod*:
$(:a,\ b:) = Abs\text{-}Sprod <strictify\cdot(\Lambda\ b.\ a)\cdot b,\ strictify\cdot(\Lambda\ a.\ b)\cdot a>$
⟨*proof*⟩

lemma *Exh-Sprod2*:
$z = \bot \lor (\exists\, a\ b.\ z = (:a,\ b:) \land a \neq \bot \land b \neq \bot)$
⟨*proof*⟩

lemma *sprodE*:
$[\![p = \bot \Longrightarrow Q;\ \bigwedge x\ y.\ [\![p = (:x,\ y:);\ x \neq \bot;\ y \neq \bot]\!] \Longrightarrow Q]\!] \Longrightarrow Q$
⟨*proof*⟩

9.4 Properties of *spair*

lemma *spair-strict1* [*simp*]: $(:\bot,\ y:) = \bot$
⟨*proof*⟩

lemma *spair-strict2* [*simp*]: $(:x,\ \bot:) = \bot$
⟨*proof*⟩

lemma *spair-strict*: $x = \bot \lor y = \bot \Longrightarrow (:x,\ y:) = \bot$
⟨*proof*⟩

lemma *spair-strict-rev*: $(:x,\ y:) \neq \bot \Longrightarrow x \neq \bot \land y \neq \bot$
⟨*proof*⟩

lemma *spair-defined* [*simp*]:
$[\![x \neq \bot;\ y \neq \bot]\!] \Longrightarrow (:x,\ y:) \neq \bot$
⟨*proof*⟩

lemma *spair-defined-rev*: $(:x,\ y:) = \bot \Longrightarrow x = \bot \lor y = \bot$
⟨*proof*⟩

lemma *spair-eq*:
$[\![x \neq \bot;\ y \neq \bot]\!] \Longrightarrow ((:x,\ y:) = (:a,\ b:)) = (x = a \land y = b)$
⟨*proof*⟩

lemma *spair-inject*:
$[\![x \neq \bot;\ y \neq \bot;\ (:x,\ y:) = (:a,\ b:)]\!] \Longrightarrow x = a \land y = b$
⟨*proof*⟩

lemma *inst-sprod-pcpo2*: $UU = (:UU, UU:)$

⟨*proof*⟩

9.5 Properties of *sfst* and *ssnd*

lemma *sfst-strict* [*simp*]: $sfst{\cdot}\bot = \bot$
⟨*proof*⟩

lemma *ssnd-strict* [*simp*]: $ssnd{\cdot}\bot = \bot$
⟨*proof*⟩

lemma *Rep-Sprod-spair*:
$Rep\text{-}Sprod\ (:a,\ b:) = {<}strictify{\cdot}(\Lambda\ b.\ a){\cdot}b,\ strictify{\cdot}(\Lambda\ a.\ b){\cdot}a{>}$
⟨*proof*⟩

lemma *sfst-spair* [*simp*]: $y \neq \bot \Longrightarrow sfst{\cdot}(:x,\ y:) = x$
⟨*proof*⟩

lemma *ssnd-spair* [*simp*]: $x \neq \bot \Longrightarrow ssnd{\cdot}(:x,\ y:) = y$
⟨*proof*⟩

lemma *sfst-defined-iff* [*simp*]: $(sfst{\cdot}p = \bot) = (p = \bot)$
⟨*proof*⟩

lemma *ssnd-defined-iff* [*simp*]: $(ssnd{\cdot}p = \bot) = (p = \bot)$
⟨*proof*⟩

lemma *sfst-defined*: $p \neq \bot \Longrightarrow sfst{\cdot}p \neq \bot$
⟨*proof*⟩

lemma *ssnd-defined*: $p \neq \bot \Longrightarrow ssnd{\cdot}p \neq \bot$
⟨*proof*⟩

lemma *surjective-pairing-Sprod2*: $(:sfst{\cdot}p,\ ssnd{\cdot}p:) = p$
⟨*proof*⟩

lemma *less-sprod*: $x \sqsubseteq y = (sfst{\cdot}x \sqsubseteq sfst{\cdot}y \land ssnd{\cdot}x \sqsubseteq ssnd{\cdot}y)$
⟨*proof*⟩

lemma *eq-sprod*: $(x = y) = (sfst{\cdot}x = sfst{\cdot}y \land ssnd{\cdot}x = ssnd{\cdot}y)$
⟨*proof*⟩

lemma *spair-less*:
$[\![x \neq \bot;\ y \neq \bot]\!] \Longrightarrow (:x,\ y:) \sqsubseteq (:a,\ b:) = (x \sqsubseteq a \land y \sqsubseteq b)$
⟨*proof*⟩

9.6 Properties of *ssplit*

lemma *ssplit1* [*simp*]: $ssplit{\cdot}f{\cdot}\bot = \bot$
⟨*proof*⟩

lemma *ssplit2* [*simp*]: $[\![x \neq \bot;\ y \neq \bot]\!] \Longrightarrow ssplit \cdot f \cdot (:x,\ y:) = f \cdot x \cdot y$
⟨*proof*⟩

lemma *ssplit3* [*simp*]: $ssplit \cdot spair \cdot z = z$
⟨*proof*⟩

end

10 Ssum: The type of strict sums

theory *Ssum*
imports *Cprod*
begin

defaultsort *pcpo*

10.1 Definition of strict sum type

pcpodef (*Ssum*) $('a,\ 'b)$ ++ (**infixr** *10*) =
$\{p{::}'a \times 'b.\ cfst \cdot p = \bot \lor csnd \cdot p = \bot\}$
⟨*proof*⟩

syntax (*xsymbols*)
++ :: [*type*, *type*] => *type* ((- ⊕/ -) [*21*, *20*] *20*)
syntax (*HTML* **output**)
++ :: [*type*, *type*] => *type* ((- ⊕/ -) [*21*, *20*] *20*)

10.2 Definitions of constructors

constdefs
$sinl :: 'a \to ('a\ ++\ 'b)$
$sinl \equiv \Lambda\ a.\ Abs\text{-}Ssum\ {<}a,\ \bot{>}$

$sinr :: 'b \to ('a\ ++\ 'b)$
$sinr \equiv \Lambda\ b.\ Abs\text{-}Ssum\ {<}\bot,\ b{>}$

10.3 Properties of *sinl* and *sinr*

lemma *sinl-Abs-Ssum*: $sinl \cdot a = Abs\text{-}Ssum\ {<}a,\ \bot{>}$
⟨*proof*⟩

lemma *sinr-Abs-Ssum*: $sinr \cdot b = Abs\text{-}Ssum\ {<}\bot,\ b{>}$
⟨*proof*⟩

lemma *Rep-Ssum-sinl*: $Rep\text{-}Ssum\ (sinl \cdot a) = {<}a,\ \bot{>}$
⟨*proof*⟩

lemma *Rep-Ssum-sinr*: $Rep\text{-}Ssum\ (sinr \cdot b) = {<}\bot,\ b{>}$

⟨*proof*⟩

lemma *sinl-strict* [*simp*]: $sinl\cdot\bot = \bot$
⟨*proof*⟩

lemma *sinr-strict* [*simp*]: $sinr\cdot\bot = \bot$
⟨*proof*⟩

lemma *sinl-eq* [*simp*]: $(sinl\cdot x = sinl\cdot y) = (x = y)$
⟨*proof*⟩

lemma *sinr-eq* [*simp*]: $(sinr\cdot x = sinr\cdot y) = (x = y)$
⟨*proof*⟩

lemma *sinl-inject*: $sinl\cdot x = sinl\cdot y \Longrightarrow x = y$
⟨*proof*⟩

lemma *sinr-inject*: $sinr\cdot x = sinr\cdot y \Longrightarrow x = y$
⟨*proof*⟩

lemma *sinl-defined-iff* [*simp*]: $(sinl\cdot x = \bot) = (x = \bot)$
⟨*proof*⟩

lemma *sinr-defined-iff* [*simp*]: $(sinr\cdot x = \bot) = (x = \bot)$
⟨*proof*⟩

lemma *sinl-defined* [*intro*!]: $x \neq \bot \Longrightarrow sinl\cdot x \neq \bot$
⟨*proof*⟩

lemma *sinr-defined* [*intro*!]: $x \neq \bot \Longrightarrow sinr\cdot x \neq \bot$
⟨*proof*⟩

10.4 Case analysis

lemma *Exh-Ssum*:
$z = \bot \lor (\exists\, a.\ z = sinl\cdot a \land a \neq \bot) \lor (\exists\, b.\ z = sinr\cdot b \land b \neq \bot)$
⟨*proof*⟩

lemma *ssumE*:
$[\![p = \bot \Longrightarrow Q;$
$\bigwedge x.\ [\![p = sinl\cdot x;\ x \neq \bot]\!] \Longrightarrow Q;$
$\bigwedge y.\ [\![p = sinr\cdot y;\ y \neq \bot]\!] \Longrightarrow Q]\!] \Longrightarrow Q$
⟨*proof*⟩

lemma *ssumE2*:
$[\![\bigwedge x.\ p = sinl\cdot x \Longrightarrow Q;\ \bigwedge y.\ p = sinr\cdot y \Longrightarrow Q]\!] \Longrightarrow Q$
⟨*proof*⟩

10.5 Ordering properties of *sinl* and *sinr*

lemma *sinl-less* [*simp*]: $(sinl\cdot x \sqsubseteq sinl\cdot y) = (x \sqsubseteq y)$
⟨*proof*⟩

lemma *sinr-less* [*simp*]: $(sinr\cdot x \sqsubseteq sinr\cdot y) = (x \sqsubseteq y)$
⟨*proof*⟩

lemma *sinl-less-sinr* [*simp*]: $(sinl\cdot x \sqsubseteq sinr\cdot y) = (x = \bot)$
⟨*proof*⟩

lemma *sinr-less-sinl* [*simp*]: $(sinr\cdot x \sqsubseteq sinl\cdot y) = (x = \bot)$
⟨*proof*⟩

lemma *sinl-eq-sinr* [*simp*]: $(sinl\cdot x = sinr\cdot y) = (x = \bot \land y = \bot)$
⟨*proof*⟩

lemma *sinr-eq-sinl* [*simp*]: $(sinr\cdot x = sinl\cdot y) = (x = \bot \land y = \bot)$
⟨*proof*⟩

10.6 Chains of strict sums

lemma *less-sinlD*: $p \sqsubseteq sinl\cdot x \Longrightarrow \exists\, y.\ p = sinl\cdot y \land y \sqsubseteq x$
⟨*proof*⟩

lemma *less-sinrD*: $p \sqsubseteq sinr\cdot x \Longrightarrow \exists\, y.\ p = sinr\cdot y \land y \sqsubseteq x$
⟨*proof*⟩

lemma *ssum-chain-lemma*:
$chain\ Y \Longrightarrow (\exists\, A.\ chain\ A \land Y = (\lambda i.\ sinl\cdot(A\ i))) \lor$
$(\exists\, B.\ chain\ B \land Y = (\lambda i.\ sinr\cdot(B\ i)))$
⟨*proof*⟩

10.7 Definitions of constants

constdefs
$Iwhen :: [{'a} \to {'c},\ {'b} \to {'c},\ {'a} \mathrel{++} {'b}] \Rightarrow {'c}$
$Iwhen \equiv \lambda f\ g\ s.$
$if\ cfst\cdot(Rep\text{-}Ssum\ s) \neq \bot\ then\ f\cdot(cfst\cdot(Rep\text{-}Ssum\ s))\ else$
$if\ csnd\cdot(Rep\text{-}Ssum\ s) \neq \bot\ then\ g\cdot(csnd\cdot(Rep\text{-}Ssum\ s))\ else\ \bot$

rewrites for *Iwhen*

lemma *Iwhen1* [*simp*]: $Iwhen\ f\ g\ \bot = \bot$
⟨*proof*⟩

lemma *Iwhen2* [*simp*]: $x \neq \bot \Longrightarrow Iwhen\ f\ g\ (sinl\cdot x) = f\cdot x$
⟨*proof*⟩

lemma *Iwhen3* [*simp*]: $y \neq \bot \Longrightarrow Iwhen\ f\ g\ (sinr\cdot y) = g\cdot y$
⟨*proof*⟩

lemma *Iwhen4*: $Iwhen\ f\ g\ (sinl{\cdot}x) = strictify{\cdot}f{\cdot}x$
⟨*proof*⟩

lemma *Iwhen5*: $Iwhen\ f\ g\ (sinr{\cdot}y) = strictify{\cdot}g{\cdot}y$
⟨*proof*⟩

10.8 Continuity of *Iwhen*

Iwhen is continuous in all arguments

lemma *cont-Iwhen1*: $cont\ (\lambda f.\ Iwhen\ f\ g\ s)$
⟨*proof*⟩

lemma *cont-Iwhen2*: $cont\ (\lambda g.\ Iwhen\ f\ g\ s)$
⟨*proof*⟩

lemma *cont-Iwhen3*: $cont\ (\lambda s.\ Iwhen\ f\ g\ s)$
⟨*proof*⟩

10.9 Continuous versions of constants

constdefs
$sscase :: ('a \to 'c) \to ('b \to 'c) \to ('a \mathbin{++} 'b) \to 'c$
$sscase \equiv \Lambda\ f\ g\ s.\ Iwhen\ f\ g\ s$

translations
case s of sinl$x => t1 | sinr$y => t2 == sscase$(LAM x. t1)$(LAM y. t2)$s

continuous versions of lemmas for *sscase*

lemma *beta-sscase*: $sscase{\cdot}f{\cdot}g{\cdot}s = Iwhen\ f\ g\ s$
⟨*proof*⟩

lemma *sscase1* [*simp*]: $sscase{\cdot}f{\cdot}g{\cdot}\bot = \bot$
⟨*proof*⟩

lemma *sscase2* [*simp*]: $x \neq \bot \Longrightarrow sscase{\cdot}f{\cdot}g{\cdot}(sinl{\cdot}x) = f{\cdot}x$
⟨*proof*⟩

lemma *sscase3* [*simp*]: $y \neq \bot \Longrightarrow sscase{\cdot}f{\cdot}g{\cdot}(sinr{\cdot}y) = g{\cdot}y$
⟨*proof*⟩

lemma *sscase4* [*simp*]: $sscase{\cdot}sinl{\cdot}sinr{\cdot}z = z$
⟨*proof*⟩

end

11 Up: The type of lifted values

theory *Up*
imports *Cfun Sum-Type Datatype*
begin

defaultsort *cpo*

11.1 Definition of new type for lifting

datatype $'a\ u = Ibottom \mid Iup\ 'a$

consts
$Ifup :: ('a \to 'b{::}pcpo) \Rightarrow 'a\ u \Rightarrow 'b$

primrec
$Ifup\ f\ Ibottom = \bot$
$Ifup\ f\ (Iup\ x) = f{\cdot}x$

11.2 Ordering on type $'a\ u$

instance *u* :: (*sq-ord*) *sq-ord* ⟨*proof*⟩

defs (**overloaded**)
less-up-def:
$(op \sqsubseteq) \equiv (\lambda x\ y.\ case\ x\ of\ Ibottom \Rightarrow True \mid Iup\ a \Rightarrow$
$(case\ y\ of\ Ibottom \Rightarrow False \mid Iup\ b \Rightarrow a \sqsubseteq b))$

lemma *minimal-up* [*iff*]: $Ibottom \sqsubseteq z$
⟨*proof*⟩

lemma *not-Iup-less* [*iff*]: $\neg\ Iup\ x \sqsubseteq Ibottom$
⟨*proof*⟩

lemma *Iup-less* [*iff*]: $(Iup\ x \sqsubseteq Iup\ y) = (x \sqsubseteq y)$
⟨*proof*⟩

11.3 Type $'a\ u$ is a partial order

lemma *refl-less-up*: $(x{::}'a\ u) \sqsubseteq x$
⟨*proof*⟩

lemma *antisym-less-up*: $[\![(x{::}'a\ u) \sqsubseteq y;\ y \sqsubseteq x]\!] \Longrightarrow x = y$
⟨*proof*⟩

lemma *trans-less-up*: $[\![(x{::}'a\ u) \sqsubseteq y;\ y \sqsubseteq z]\!] \Longrightarrow x \sqsubseteq z$
⟨*proof*⟩

instance *u* :: (*cpo*) *po*
⟨*proof*⟩

11.4 Type $'a\ u$ is a cpo

lemma *is-lub-Iup*:
$range\ S <<|\ x \Longrightarrow range\ (\lambda i.\ Iup\ (S\ i)) <<|\ Iup\ x$
⟨*proof*⟩

Now some lemmas about chains of $'a\ u$ elements

lemma *up-lemma1*: $z \neq Ibottom \Longrightarrow Iup\ (THE\ a.\ Iup\ a = z) = z$
⟨*proof*⟩

lemma *up-lemma2*:
$[\![chain\ Y;\ Y\ j \neq Ibottom]\!] \Longrightarrow Y\ (i + j) \neq Ibottom$
⟨*proof*⟩

lemma *up-lemma3*:
$[\![chain\ Y;\ Y\ j \neq Ibottom]\!] \Longrightarrow Iup\ (THE\ a.\ Iup\ a = Y\ (i + j)) = Y\ (i + j)$
⟨*proof*⟩

lemma *up-lemma4*:
$[\![chain\ Y;\ Y\ j \neq Ibottom]\!] \Longrightarrow chain\ (\lambda i.\ THE\ a.\ Iup\ a = Y\ (i + j))$
⟨*proof*⟩

lemma *up-lemma5*:
$[\![chain\ Y;\ Y\ j \neq Ibottom]\!] \Longrightarrow$
$(\lambda i.\ Y\ (i + j)) = (\lambda i.\ Iup\ (THE\ a.\ Iup\ a = Y\ (i + j)))$
⟨*proof*⟩

lemma *up-lemma6*:
$[\![chain\ Y;\ Y\ j \neq Ibottom]\!]$
$\Longrightarrow range\ Y <<|\ Iup\ (\bigsqcup i.\ THE\ a.\ Iup\ a = Y(i + j))$
⟨*proof*⟩

lemma *up-chain-cases*:
$chain\ Y \Longrightarrow$
$(\exists A.\ chain\ A \wedge lub\ (range\ Y) = Iup\ (lub\ (range\ A)) \wedge$
$(\exists j.\ \forall i.\ Y\ (i + j) = Iup\ (A\ i))) \vee (Y = (\lambda i.\ Ibottom))$
⟨*proof*⟩

lemma *cpo-up*: $chain\ (Y{::}nat \Rightarrow 'a\ u) \Longrightarrow \exists x.\ range\ Y <<|\ x$
⟨*proof*⟩

instance $u :: (cpo)\ cpo$
⟨*proof*⟩

11.5 Type $'a\ u$ is pointed

lemma *least-up*: $\exists x{::}'a\ u.\ \forall y.\ x \sqsubseteq y$
⟨*proof*⟩

instance $u :: (cpo)\ pcpo$

⟨*proof*⟩

for compatibility with old HOLCF-Version

lemma *inst-up-pcpo*: $\perp = Ibottom$
⟨*proof*⟩

11.6 Continuity of *Iup* and *Ifup*

continuity for *Iup*

lemma *cont-Iup*: *cont Iup*
⟨*proof*⟩

continuity for *Ifup*

lemma *cont-Ifup1*: $cont\ (\lambda f.\ Ifup\ f\ x)$
⟨*proof*⟩

lemma *monofun-Ifup2*: $monofun\ (\lambda x.\ Ifup\ f\ x)$
⟨*proof*⟩

lemma *cont-Ifup2*: $cont\ (\lambda x.\ Ifup\ f\ x)$
⟨*proof*⟩

11.7 Continuous versions of constants

constdefs
  $up\ ::\ 'a \to 'a\ u$
  $up \equiv \Lambda\ x.\ Iup\ x$

  $fup :: ('a \to 'b::pcpo) \to 'a\ u \to 'b$
  $fup \equiv \Lambda\ f\ p.\ Ifup\ f\ p$

translations
$case\ l\ of\ up{\cdot}x \Rightarrow t == fup{\cdot}(LAM\ x.\ t){\cdot}l$

continuous versions of lemmas for $'a\ u$

lemma *Exh-Up*: $z = \perp \vee (\exists\, x.\ z = up{\cdot}x)$
⟨*proof*⟩

lemma *up-eq* [*simp*]: $(up{\cdot}x = up{\cdot}y) = (x = y)$
⟨*proof*⟩

lemma *up-inject*: $up{\cdot}x = up{\cdot}y \Longrightarrow x = y$
⟨*proof*⟩

lemma *up-defined* [*simp*]: $up{\cdot}x \neq \perp$
⟨*proof*⟩

lemma *not-up-less-UU* [*simp*]: $\neg\ up{\cdot}x \sqsubseteq \perp$

⟨*proof*⟩

lemma *up-less* [*simp*]: $(up{\cdot}x \sqsubseteq up{\cdot}y) = (x \sqsubseteq y)$
⟨*proof*⟩

lemma *upE*: $[\![p = \bot \Longrightarrow Q; \bigwedge x.\ p = up{\cdot}x \Longrightarrow Q]\!] \Longrightarrow Q$
⟨*proof*⟩

lemma *fup1* [*simp*]: $fup{\cdot}f{\cdot}\bot = \bot$
⟨*proof*⟩

lemma *fup2* [*simp*]: $fup{\cdot}f{\cdot}(up{\cdot}x) = f{\cdot}x$
⟨*proof*⟩

lemma *fup3* [*simp*]: $fup{\cdot}up{\cdot}x = x$
⟨*proof*⟩

end

12 Discrete: Discrete cpo types

theory *Discrete*
imports *Cont Datatype*
begin

datatype *'a discr = Discr 'a :: type*

12.1 Type *'a discr* is a partial order

instance *discr* :: (*type*) *sq-ord* ⟨*proof*⟩

defs (overloaded)
less-discr-def: ((*op* <<)::('*a*::*type*)*discr*=>'*a discr*=>*bool*) == *op* =

lemma *discr-less-eq* [*iff*]: ((*x*::('*a*::*type*)*discr*) << *y*) = (*x* = *y*)
⟨*proof*⟩

instance *discr* :: (*type*) *po*
⟨*proof*⟩

12.2 Type *'a discr* is a cpo

lemma *discr-chain0*:
!!*S*::*nat*=>('*a*::*type*)*discr*. *chain S* ==> *S i* = *S 0*
⟨*proof*⟩

lemma *discr-chain-range0* [*simp*]:
!!*S*::*nat*=>('*a*::*type*)*discr*. *chain*(*S*) ==> *range*(*S*) = {*S 0*}

⟨*proof*⟩

lemma *discr-cpo*:
!!*S*. *chain S* ==> ? *x*::('*a*::*type*)*discr*. *range*(*S*) <<| *x*
⟨*proof*⟩

instance *discr* :: (*type*) *cpo*
⟨*proof*⟩

12.3 *undiscr*

constdefs
undiscr :: ('*a*::*type*)*discr* => '*a*
undiscr x == (*case x of Discr y* => *y*)

lemma *undiscr-Discr* [*simp*]: *undiscr*(*Discr x*) = *x*
⟨*proof*⟩

lemma *discr-chain-f-range0*:
!!*S*::*nat*=>('*a*::*type*)*discr*. *chain*(*S*) ==> *range*(%*i*. *f*(*S i*)) = {*f*(*S 0*)}
⟨*proof*⟩

lemma *cont-discr* [*iff*]: *cont*(%*x*::('*a*::*type*)*discr*. *f x*)
⟨*proof*⟩

end

13 Lift: Lifting types of class type to flat pcpo's

theory *Lift*
imports *Discrete Up Cprod*
begin

defaultsort *type*

pcpodef '*a lift* = *UNIV* :: '*a discr u set*
⟨*proof*⟩

lemmas *inst-lift-pcpo* = *Abs-lift-strict* [*symmetric*]

constdefs
Def :: '*a* ⇒ '*a lift*
Def x ≡ *Abs-lift* (*up*·(*Discr x*))

13.1 Lift as a datatype

lemma *lift-distinct1*: ⊥ ≠ *Def x*
⟨*proof*⟩

lemma *lift-distinct2*: $Def\ x \neq \bot$
⟨*proof*⟩

lemma *Def-inject*: $(Def\ x = Def\ y) = (x = y)$
⟨*proof*⟩

lemma *lift-induct*: $[\![P\ \bot;\ \bigwedge x.\ P\ (Def\ x)]\!] \Longrightarrow P\ y$
⟨*proof*⟩

rep-datatype *lift*
distinct *lift-distinct1 lift-distinct2*
inject *Def-inject*
induction *lift-induct*

lemma *Def-not-UU*: $Def\ a \neq UU$
⟨*proof*⟩

$\bot$ and *Def*

lemma *Lift-exhaust*: $x = \bot \vee (\exists\, y.\ x = Def\ y)$
⟨*proof*⟩

lemma *Lift-cases*: $[\![x = \bot \Longrightarrow P;\ \exists\, a.\ x = Def\ a \Longrightarrow P]\!] \Longrightarrow P$
⟨*proof*⟩

lemma *not-Undef-is-Def*: $(x \neq \bot) = (\exists\, y.\ x = Def\ y)$
⟨*proof*⟩

lemma *lift-definedE*: $[\![x \neq \bot;\ \bigwedge a.\ x = Def\ a \Longrightarrow R]\!] \Longrightarrow R$
⟨*proof*⟩

For $x \neq \bot$ in assumptions *def-tac* replaces x by *Def a* in conclusion.

⟨*ML*⟩

lemma *DefE*: $Def\ x = \bot \Longrightarrow R$
⟨*proof*⟩

lemma *DefE2*: $[\![x = Def\ s;\ x = \bot]\!] \Longrightarrow R$
⟨*proof*⟩

lemma *Def-inject-less-eq*: $Def\ x \sqsubseteq Def\ y = (x = y)$
⟨*proof*⟩

lemma *Def-less-is-eq* [*simp*]: $Def\ x \sqsubseteq y = (Def\ x = y)$
⟨*proof*⟩

13.2 Lift is flat

lemma *less-lift*: $(x{::}'a\ lift) \sqsubseteq y = (x = y \vee x = \bot)$

⟨*proof*⟩

instance *lift* :: (*type*) *flat*
⟨*proof*⟩

Two specific lemmas for the combination of LCF and HOL terms.

lemma *cont-Rep-CFun-app*: $[\![cont\ g;\ cont\ f]\!] \Longrightarrow cont(\lambda x.\ ((f\ x)\cdot(g\ x))\ s)$
⟨*proof*⟩

lemma *cont-Rep-CFun-app-app*: $[\![cont\ g;\ cont\ f]\!] \Longrightarrow cont(\lambda x.\ ((f\ x)\cdot(g\ x))\ s\ t)$
⟨*proof*⟩

13.3 Further operations

constdefs
$flift1 :: ('a \Rightarrow 'b{::}pcpo) \Rightarrow ('a\ lift \rightarrow 'b)$ (**binder** *FLIFT* *10*)
$flift1 \equiv \lambda f.\ (\Lambda\ x.\ lift\text{-}case\ \bot\ f\ x)$

$flift2 :: ('a \Rightarrow 'b) \Rightarrow ('a\ lift \rightarrow 'b\ lift)$
$flift2\ f \equiv FLIFT\ x.\ Def\ (f\ x)$

$liftpair :: 'a\ lift \times 'b\ lift \Rightarrow ('a \times 'b)\ lift$
$liftpair\ x \equiv csplit\cdot(FLIFT\ x\ y.\ Def\ (x,\ y))\cdot x$

13.4 Continuity Proofs for flift1, flift2

Need the instance of *flat*.

lemma *cont-lift-case1*: $cont\ (\lambda f.\ lift\text{-}case\ a\ f\ x)$
⟨*proof*⟩

lemma *cont-lift-case2*: $cont\ (\lambda x.\ lift\text{-}case\ \bot\ f\ x)$
⟨*proof*⟩

lemma *cont-flift1*: *cont flift1*
⟨*proof*⟩

lemma *cont2cont-flift1*:
$[\![\bigwedge y.\ cont\ (\lambda x.\ f\ x\ y)]\!] \Longrightarrow cont\ (\lambda x.\ FLIFT\ y.\ f\ x\ y)$
⟨*proof*⟩

lemma *cont2cont-lift-case*:
$[\![\bigwedge y.\ cont\ (\lambda x.\ f\ x\ y);\ cont\ g]\!] \Longrightarrow cont\ (\lambda x.\ lift\text{-}case\ UU\ (f\ x)\ (g\ x))$
⟨*proof*⟩

rewrites for *flift1*, *flift2*

lemma *flift1-Def* [*simp*]: $flift1\ f\cdot(Def\ x) = (f\ x)$
⟨*proof*⟩

lemma *flift2-Def* [*simp*]: *flift2* $f \cdot (Def\ x) = Def\ (f\ x)$
⟨*proof*⟩

lemma *flift1-strict* [*simp*]: *flift1* $f \cdot \bot = \bot$
⟨*proof*⟩

lemma *flift2-strict* [*simp*]: *flift2* $f \cdot \bot = \bot$
⟨*proof*⟩

lemma *flift2-defined* [*simp*]: $x \neq \bot \Longrightarrow (flift2\ f) \cdot x \neq \bot$
⟨*proof*⟩

Extension of *cont-tac* and installation of simplifier.

lemmas *cont-lemmas-ext* [*simp*] =
cont2cont-flift1 cont2cont-lift-case cont2cont-lambda
cont-Rep-CFun-app cont-Rep-CFun-app-app cont-if

⟨*ML*⟩

end

14 One: The unit domain

theory *One*
imports *Lift*
begin

types *one* = *unit lift*

constdefs
ONE :: *one*
$ONE \equiv Def\ ()$

translations
one <= (*type*) *unit lift*

Exhaustion and Elimination for type *one*

lemma *Exh-one*: $t = \bot \vee t = ONE$
⟨*proof*⟩

lemma *oneE*: $[\![p = \bot \Longrightarrow Q;\ p = ONE \Longrightarrow Q]\!] \Longrightarrow Q$
⟨*proof*⟩

lemma *dist-less-one* [*simp*]: $\neg\ ONE \sqsubseteq \bot$
⟨*proof*⟩

lemma *dist-eq-one* [*simp*]: $ONE \neq \bot\ \bot \neq ONE$

⟨*proof*⟩

end

15 Tr: The type of lifted booleans

theory *Tr*
imports *Lift*
begin

defaultsort *pcpo*

types
tr = *bool lift*

translations
tr <= (*type*) *bool lift*

consts

```
TT        :: tr
FF        :: tr
Icifte    :: tr -> 'c -> 'c -> 'c
trand     :: tr -> tr -> tr
tror      :: tr -> tr -> tr
neg       :: tr -> tr
If2       :: tr=>'c=>'c=>'c
```

syntax

```
@cifte    :: tr=>'c=>'c=>'c ((3If -/ (then -/ else -) fi) 60)
@andalso  :: tr => tr => tr (- andalso - [36,35] 35)
@orelse   :: tr => tr => tr (- orelse -  [31,30] 30)
```

translations

```
x andalso y == trand$x$y
x orelse y  == tror$x$y
If b then e1 else e2 fi == Icifte$b$e1$e2
```

defs

```
TT-def:      TT==Def True
FF-def:      FF==Def False
neg-def:     neg == flift2 Not
ifte-def:    Icifte == (LAM b t e. flift1(%b. if b then t else e)$b)
andalso-def: trand == (LAM x y. If x then y else FF fi)
orelse-def:  tror == (LAM x y. If x then TT else y fi)
If2-def:     If2 Q x y == If Q then x else y fi
```

Exhaustion and Elimination for type *tr*

lemma *Exh-tr*: *t=UU* | *t* = *TT* | *t* = *FF*
⟨*proof*⟩

lemma *trE*: [| $p{=}UU$ ==> Q; $p = TT$ ==>Q; $p = FF$ ==>Q|] ==>Q
⟨*proof*⟩

tactic for tr-thms with case split

lemmas *tr-defs* = *andalso-def orelse-def neg-def ifte-def TT-def FF-def*

distinctness for type *tr*

lemma *dist-less-tr* [*simp*]: ~*TT* << *UU* ~*FF* << *UU* ~*TT* << *FF* ~*FF* << *TT*
⟨*proof*⟩

lemma *dist-eq-tr* [*simp*]: *TT*~=*UU* *FF*~=*UU* *TT*~=*FF* *UU*~=*TT* *UU*~=*FF* *FF*~=*TT*
⟨*proof*⟩

lemmas about andalso, orelse, neg and if

lemma *ifte-thms* [*simp*]:
  If UU then e1 else e2 fi = *UU*
  If FF then e1 else e2 fi = *e2*
  If TT then e1 else e2 fi = *e1*
⟨*proof*⟩

lemma *andalso-thms* [*simp*]:
  (*TT andalso y*) = *y*
  (*FF andalso y*) = *FF*
  (*UU andalso y*) = *UU*
  (*y andalso TT*) = *y*
  (*y andalso y*) = *y*
⟨*proof*⟩

lemma *orelse-thms* [*simp*]:
  (*TT orelse y*) = *TT*
  (*FF orelse y*) = *y*
  (*UU orelse y*) = *UU*
  (*y orelse FF*) = *y*
  (*y orelse y*) = *y*
⟨*proof*⟩

lemma *neg-thms* [*simp*]:
  neg$*TT* = *FF*
  neg$*FF* = *TT*
  neg$*UU* = *UU*
⟨*proof*⟩

split-tac for If via If2 because the constant has to be a constant

lemma *split-If2*:
  P (*If2 Q x y*) = ((*Q*=*UU* --> *P UU*) & (*Q*=*TT* --> *P x*) & (*Q*=*FF* --> *P y*))

⟨*proof*⟩

⟨*ML*⟩

15.1 Rewriting of HOLCF operations to HOL functions

lemma *andalso-or*:
$!!t.[|t\sim=UU|]==> ((t\ andalso\ s)=FF)=(t=FF\ |\ s=FF)$
⟨*proof*⟩

lemma *andalso-and*: $[|t\sim=UU|]==> ((t\ andalso\ s)\sim=FF)=(t\sim=FF\ \&\ s\sim=FF)$
⟨*proof*⟩

lemma *Def-bool1* [*simp*]: $(Def\ x\ \sim= FF) = x$
⟨*proof*⟩

lemma *Def-bool2* [*simp*]: $(Def\ x = FF) = (\sim x)$
⟨*proof*⟩

lemma *Def-bool3* [*simp*]: $(Def\ x = TT) = x$
⟨*proof*⟩

lemma *Def-bool4* [*simp*]: $(Def\ x\ \sim= TT) = (\sim x)$
⟨*proof*⟩

lemma *If-and-if*:
  $(If\ Def\ P\ then\ A\ else\ B\ fi)= (if\ P\ then\ A\ else\ B)$
⟨*proof*⟩

15.2 admissibility

The following rewrite rules for admissibility should in the future be replaced by a more general admissibility test that also checks chain-finiteness, of which these lemmata are specific examples

lemma *adm-trick-1*: $(x\sim=FF) = (x=TT|x=UU)$
⟨*proof*⟩

lemma *adm-trick-2*: $(x\sim=TT) = (x=FF|x=UU)$
⟨*proof*⟩

lemmas *adm-tricks = adm-trick-1 adm-trick-2*

lemma *adm-nTT* [*simp*]: $cont(f) ==> adm\ (\%x.\ (f\ x)\sim=TT)$
⟨*proof*⟩

lemma *adm-nFF* [*simp*]: $cont(f) ==> adm\ (\%x.\ (f\ x)\sim=FF)$
⟨*proof*⟩

end

16 Fix: Fixed point operator and admissibility

theory *Fix*
imports *Cfun Cprod Adm*
begin

defaultsort *pcpo*

16.1 Definitions

consts
$iterate :: nat \Rightarrow ('a \rightarrow 'a) \Rightarrow 'a \Rightarrow 'a$
$Ifix\ :: ('a \rightarrow 'a) \Rightarrow 'a$
$fix\ :: ('a \rightarrow 'a) \rightarrow 'a$
$admw\ :: ('a \Rightarrow bool) \Rightarrow bool$

primrec
$iterate\text{-}0$: $iterate\ 0\ F\ x = x$
$iterate\text{-}Suc$: $iterate\ (Suc\ n)\ F\ x = F \cdot (iterate\ n\ F\ x)$

defs
$Ifix\text{-}def$: $Ifix \equiv \lambda F.\ \bigsqcup i.\ iterate\ i\ F\ \bot$
$fix\text{-}def$: $fix \equiv \Lambda\ F.\ Ifix\ F$

$admw\text{-}def$: $admw\ P \equiv \forall F.\ (\forall n.\ P\ (iterate\ n\ F\ \bot)) \longrightarrow$
$P\ (\bigsqcup i.\ iterate\ i\ F\ \bot)$

16.2 Binder syntax for *fix*

syntax
@*FIX* :: ('a => 'a) => 'a (**binder** *FIX* 10)
@*FIXP* :: [*patterns*, 'a] => 'a ((*3FIX* <->./ -) [0, 10] 10)

syntax (*xsymbols*)
FIX :: [*idt*, 'a] => 'a ((3μ-./ -) [0, 10] 10)
@*FIXP* :: [*patterns*, 'a] => 'a ((3μ()<->./ -) [0, 10] 10)

translations
FIX x. LAM y. t == *fix*·(*LAM x y. t*)
FIX x. t == *fix*·(*LAM x. t*)
FIX <xs>. *t* == *fix*·(*LAM* <xs>. *t*)

16.3 Properties of *iterate* and *fix*

derive inductive properties of iterate from primitive recursion

lemma *iterate-Suc2*: $iterate\ (Suc\ n)\ F\ x = iterate\ n\ F\ (F \cdot x)$

⟨*proof*⟩

The sequence of function iterations is a chain. This property is essential since monotonicity of iterate makes no sense.

lemma *chain-iterate2*: $x \sqsubseteq F{\cdot}x \Longrightarrow chain\ (\lambda i.\ iterate\ i\ F\ x)$
⟨*proof*⟩

lemma *chain-iterate*: $chain\ (\lambda i.\ iterate\ i\ F\ \bot)$
⟨*proof*⟩

Kleene's fixed point theorems for continuous functions in pointed omega cpo's

lemma *Ifix-eq*: $Ifix\ F = F{\cdot}(Ifix\ F)$
⟨*proof*⟩

lemma *Ifix-least*: $F{\cdot}x = x \Longrightarrow Ifix\ F \sqsubseteq x$
⟨*proof*⟩

continuity of *iterate*

lemma *cont-iterate1*: $cont\ (\lambda F.\ iterate\ n\ F\ x)$
⟨*proof*⟩

lemma *cont-iterate2*: $cont\ (\lambda x.\ iterate\ n\ F\ x)$
⟨*proof*⟩

lemma *cont-iterate*: $cont\ (iterate\ n)$
⟨*proof*⟩

lemmas *monofun-iterate2* = *cont-iterate2* [*THEN cont2mono*, *standard*]
lemmas *contlub-iterate2* = *cont-iterate2* [*THEN cont2contlub*, *standard*]

continuity of *Ifix*

lemma *cont-Ifix*: *cont Ifix*
⟨*proof*⟩

propagate properties of *Ifix* to its continuous counterpart

lemma *fix-eq*: $fix{\cdot}F = F{\cdot}(fix{\cdot}F)$
⟨*proof*⟩

lemma *fix-least*: $F{\cdot}x = x \Longrightarrow fix{\cdot}F \sqsubseteq x$
⟨*proof*⟩

lemma *fix-eqI*: $[\![F{\cdot}x = x;\ \forall z.\ F{\cdot}z = z \longrightarrow x \sqsubseteq z]\!] \Longrightarrow x = fix{\cdot}F$
⟨*proof*⟩

lemma *fix-eq2*: $f \equiv fix{\cdot}F \Longrightarrow f = F{\cdot}f$
⟨*proof*⟩

lemma *fix-eq3*: $f \equiv \mathit{fix}{\cdot}F \Longrightarrow f{\cdot}x = F{\cdot}f{\cdot}x$
⟨*proof*⟩

lemma *fix-eq4*: $f = \mathit{fix}{\cdot}F \Longrightarrow f = F{\cdot}f$
⟨*proof*⟩

lemma *fix-eq5*: $f = \mathit{fix}{\cdot}F \Longrightarrow f{\cdot}x = F{\cdot}f{\cdot}x$
⟨*proof*⟩

direct connection between *fix* and iteration without *Ifix*

lemma *fix-def2*: $\mathit{fix}{\cdot}F = (\bigsqcup i.\ \mathit{iterate}\ i\ F\ \bot)$
⟨*proof*⟩

strictness of *fix*

lemma *fix-defined-iff*: $(\mathit{fix}{\cdot}F = \bot) = (F{\cdot}\bot = \bot)$
⟨*proof*⟩

lemma *fix-strict*: $F{\cdot}\bot = \bot \Longrightarrow \mathit{fix}{\cdot}F = \bot$
⟨*proof*⟩

lemma *fix-defined*: $F{\cdot}\bot \neq \bot \Longrightarrow \mathit{fix}{\cdot}F \neq \bot$
⟨*proof*⟩

fix applied to identity and constant functions

lemma *fix-id*: $(\mu\ x.\ x) = \bot$
⟨*proof*⟩

lemma *fix-const*: $(\mu\ x.\ c) = c$
⟨*proof*⟩

16.4 Admissibility and fixed point induction

an admissible formula is also weak admissible

lemma *adm-impl-admw*: $\mathit{adm}\ P \Longrightarrow \mathit{admw}\ P$
⟨*proof*⟩

some lemmata for functions with flat/chfin domain/range types

lemma *adm-chfindom*: $\mathit{adm}\ (\lambda(u{::}'a{::}\mathit{cpo} \rightarrow {'b}{::}\mathit{chfin}).\ P(u{\cdot}s))$
⟨*proof*⟩

fixed point induction

lemma *fix-ind*: $[\![\mathit{adm}\ P;\ P\ \bot;\ \bigwedge x.\ P\ x \Longrightarrow P\ (F{\cdot}x)]\!] \Longrightarrow P\ (\mathit{fix}{\cdot}F)$
⟨*proof*⟩

lemma *def-fix-ind*:
$[\![f \equiv \mathit{fix}{\cdot}F;\ \mathit{adm}\ P;\ P\ \bot;\ \bigwedge x.\ P\ x \Longrightarrow P\ (F{\cdot}x)]\!] \Longrightarrow P\ f$
⟨*proof*⟩

computational induction for weak admissible formulae

lemma *wfix-ind*: $[\![admw\ P;\ \forall\, n.\ P\ (iterate\ n\ F\ \bot)]\!] \Longrightarrow P\ (fix\cdot F)$
⟨*proof*⟩

lemma *def-wfix-ind*:
$[\![f \equiv fix\cdot F;\ admw\ P;\ \forall\, n.\ P\ (iterate\ n\ F\ \bot)]\!] \Longrightarrow P\ f$
⟨*proof*⟩

end

17 Fixrec: Package for defining recursive functions in HOLCF

theory *Fixrec*
imports *Sprod Ssum Up One Tr Fix*
uses (*fixrec-package.ML*)
begin

17.1 Maybe monad type

defaultsort *cpo*

types $'a\ maybe = one\ {+}{+}\ 'a\ u$

constdefs
$fail :: 'a\ maybe$
$fail \equiv sinl\cdot ONE$

$return :: 'a \to 'a\ maybe$
$return \equiv sinr\ oo\ up$

lemma *maybeE*:
$[\![p = \bot \Longrightarrow Q;\ p = fail \Longrightarrow Q;\ \bigwedge x.\ p = return\cdot x \Longrightarrow Q]\!] \Longrightarrow Q$
⟨*proof*⟩

17.2 Monadic bind operator

constdefs
$bind :: 'a\ maybe \to ('a \to 'b\ maybe) \to 'b\ maybe$
$bind \equiv \Lambda\ m\ f.\ sscase\cdot sinl\cdot(fup\cdot f)\cdot m$

syntax
$\text{-}bind :: 'a\ maybe \Rightarrow ('a \to 'b\ maybe) \Rightarrow 'b\ maybe$
$((\text{-}\ {>}{>}{=}\ \text{-})\ [50,\ 51]\ 50)$

translations $m\ {>}{>}{=}\ k == bind\cdot m\cdot k$

nonterminals
maybebind maybebinds

syntax
-MBIND :: *pttrn* ⇒ *'a maybe* ⇒ *maybebind* ((*2*- <−/ -) *10*)
:: *maybebind* ⇒ *maybebinds* (-)

-MBINDS :: [*maybebind*, *maybebinds*] ⇒ *maybebinds* (-;/ -)
-MDO :: [*maybebinds*, *'a maybe*] ⇒ *'a maybe* ((*do* -;/ (-)) *10*)

translations
-MDO (*-MBINDS b bs*) *e* == *-MDO b* (*-MDO bs e*)
do (*x*,*y*) <− *m*; *e* == *m* >>= (*LAM* <*x*,*y*>. *e*)
do x <− *m*; *e* == *m* >>= (*LAM x*. *e*)

monad laws

lemma *bind-strict* [*simp*]: *UU* >>= *f* = *UU*
⟨*proof*⟩

lemma *bind-fail* [*simp*]: *fail* >>= *f* = *fail*
⟨*proof*⟩

lemma *left-unit* [*simp*]: (*return*·*a*) >>= *k* = *k*·*a*
⟨*proof*⟩

lemma *right-unit* [*simp*]: *m* >>= *return* = *m*
⟨*proof*⟩

lemma *bind-assoc* [*simp*]:
(*do b* <− (*do a* <− *m*; *k*·*a*); *h*·*b*) = (*do a* <− *m*; *b* <− *k*·*a*; *h*·*b*)
⟨*proof*⟩

17.3 Run operator

constdefs
run:: *'a*::*pcpo maybe* → *'a*
run ≡ *sscase*·⊥·(*fup*·*ID*)

rewrite rules for run

lemma *run-strict* [*simp*]: *run*·⊥ = ⊥
⟨*proof*⟩

lemma *run-fail* [*simp*]: *run*·*fail* = ⊥
⟨*proof*⟩

lemma *run-return* [*simp*]: *run*·(*return*·*x*) = *x*
⟨*proof*⟩

17.4 Monad plus operator

constdefs
  mplus :: $'a$ *maybe* $\rightarrow$ $'a$ *maybe* $\rightarrow$ $'a$ *maybe*
  mplus $\equiv$ Λ *m1 m2*. *sscase*·(Λ *x*. *m2*)·(*fup*·*return*)·*m1*

syntax +++ :: $'a$ *maybe* $\Rightarrow$ $'a$ *maybe* $\Rightarrow$ $'a$ *maybe* (**infixr** *65*)
translations *x* +++ *y* == *mplus*·*x*·*y*

rewrite rules for mplus

lemma *mplus-strict* [*simp*]: $\perp$ +++ *m* = $\perp$
⟨*proof*⟩

lemma *mplus-fail* [*simp*]: *fail* +++ *m* = *m*
⟨*proof*⟩

lemma *mplus-return* [*simp*]: *return*·*x* +++ *m* = *return*·*x*
⟨*proof*⟩

lemma *mplus-fail2* [*simp*]: *m* +++ *fail* = *m*
⟨*proof*⟩

lemma *mplus-assoc*: (*x* +++ *y*) +++ *z* = *x* +++ (*y* +++ *z*)
⟨*proof*⟩

17.5 Match functions for built-in types

defaultsort *pcpo*

constdefs
  match-UU :: $'a$ $\rightarrow$ *unit maybe*
  match-UU $\equiv$ Λ *x*. *fail*

  match-cpair :: $'a$::*cpo* $\times$ $'b$::*cpo* $\rightarrow$ ($'a$ $\times$ $'b$) *maybe*
  match-cpair $\equiv$ *csplit*·(Λ *x y*. *return*·<*x*,*y*>)

  match-spair :: $'a$ $\otimes$ $'b$ $\rightarrow$ ($'a$ $\times$ $'b$) *maybe*
  match-spair $\equiv$ *ssplit*·(Λ *x y*. *return*·<*x*,*y*>)

  match-sinl :: $'a$ $\oplus$ $'b$ $\rightarrow$ $'a$ *maybe*
  match-sinl $\equiv$ *sscase*·*return*·(Λ *y*. *fail*)

  match-sinr :: $'a$ $\oplus$ $'b$ $\rightarrow$ $'b$ *maybe*
  match-sinr $\equiv$ *sscase*·(Λ *x*. *fail*)·*return*

  match-up :: $'a$::*cpo u* $\rightarrow$ $'a$ *maybe*
  match-up $\equiv$ *fup*·*return*

  match-ONE :: *one* $\rightarrow$ *unit maybe*
  match-ONE $\equiv$ *flift1* (λ*u*. *return*·())

$match\text{-}TT :: tr \to unit\ maybe$
$match\text{-}TT \equiv flift1\ (\lambda b.\ if\ b\ then\ return\cdot()\ else\ fail)$

$match\text{-}FF :: tr \to unit\ maybe$
$match\text{-}FF \equiv flift1\ (\lambda b.\ if\ b\ then\ fail\ else\ return\cdot())$

lemma *match-UU-simps* [*simp*]:
$match\text{-}UU\cdot x = fail$
⟨*proof*⟩

lemma *match-cpair-simps* [*simp*]:
$match\text{-}cpair\cdot{<}x,y{>} = return\cdot{<}x,y{>}$
⟨*proof*⟩

lemma *match-spair-simps* [*simp*]:
$[\![x \neq \bot;\ y \neq \bot]\!] \Longrightarrow match\text{-}spair\cdot(:x,y:) = return\cdot{<}x,y{>}$
$match\text{-}spair\cdot\bot = \bot$
⟨*proof*⟩

lemma *match-sinl-simps* [*simp*]:
$x \neq \bot \Longrightarrow match\text{-}sinl\cdot(sinl\cdot x) = return\cdot x$
$x \neq \bot \Longrightarrow match\text{-}sinl\cdot(sinr\cdot x) = fail$
$match\text{-}sinl\cdot\bot = \bot$
⟨*proof*⟩

lemma *match-sinr-simps* [*simp*]:
$x \neq \bot \Longrightarrow match\text{-}sinr\cdot(sinr\cdot x) = return\cdot x$
$x \neq \bot \Longrightarrow match\text{-}sinr\cdot(sinl\cdot x) = fail$
$match\text{-}sinr\cdot\bot = \bot$
⟨*proof*⟩

lemma *match-up-simps* [*simp*]:
$match\text{-}up\cdot(up\cdot x) = return\cdot x$
$match\text{-}up\cdot\bot = \bot$
⟨*proof*⟩

lemma *match-ONE-simps* [*simp*]:
$match\text{-}ONE\cdot ONE = return\cdot()$
$match\text{-}ONE\cdot\bot = \bot$
⟨*proof*⟩

lemma *match-TT-simps* [*simp*]:
$match\text{-}TT\cdot TT = return\cdot()$
$match\text{-}TT\cdot FF = fail$
$match\text{-}TT\cdot\bot = \bot$
⟨*proof*⟩

lemma *match-FF-simps* [*simp*]:

  $match\text{-}FF{\cdot}FF = return{\cdot}()$
  $match\text{-}FF{\cdot}TT = fail$
  $match\text{-}FF{\cdot}\bot = \bot$
$\langle proof \rangle$

17.6 Mutual recursion

The following rules are used to prove unfolding theorems from fixed-point definitions of mutually recursive functions.

lemma *cpair-equalI*: $[\![x \equiv cfst{\cdot}p;\ y \equiv csnd{\cdot}p]\!] \Longrightarrow <x,y> \equiv p$
$\langle proof \rangle$

lemma *cpair-eqD1*: $<x,y> = <x',y'> \Longrightarrow x = x'$
$\langle proof \rangle$

lemma *cpair-eqD2*: $<x,y> = <x',y'> \Longrightarrow y = y'$
$\langle proof \rangle$

lemma for proving rewrite rules

lemma *ssubst-lhs*: $[\![t = s;\ P\ s = Q]\!] \Longrightarrow P\ t = Q$
$\langle proof \rangle$

$\langle ML \rangle$

17.7 Initializing the fixrec package

$\langle ML \rangle$

end

18 Domain: Domain package

theory *Domain*
imports *Ssum Sprod Up One Tr Fixrec*

begin

defaultsort *pcpo*

18.1 Continuous isomorphisms

A locale for continuous isomorphisms

locale *iso* =
  fixes $abs :: 'a \to 'b$
  fixes $rep :: 'b \to 'a$
  assumes *abs-iso* [*simp*]: $rep{\cdot}(abs{\cdot}x) = x$

assumes *rep-iso* [*simp*]: $abs \cdot (rep \cdot y) = y$

lemma (**in** *iso*) *swap*: *iso rep abs*
⟨*proof*⟩

lemma (**in** *iso*) *abs-strict*: $abs \cdot \bot = \bot$
⟨*proof*⟩

lemma (**in** *iso*) *rep-strict*: $rep \cdot \bot = \bot$
⟨*proof*⟩

lemma (**in** *iso*) *abs-defin'*: $abs \cdot z = \bot \Longrightarrow z = \bot$
⟨*proof*⟩

lemma (**in** *iso*) *rep-defin'*: $rep \cdot z = \bot \Longrightarrow z = \bot$
⟨*proof*⟩

lemma (**in** *iso*) *abs-defined*: $z \neq \bot \Longrightarrow abs \cdot z \neq \bot$
⟨*proof*⟩

lemma (**in** *iso*) *rep-defined*: $z \neq \bot \Longrightarrow rep \cdot z \neq \bot$
⟨*proof*⟩

lemma (**in** *iso*) *iso-swap*: $(x = abs \cdot y) = (rep \cdot x = y)$
⟨*proof*⟩

18.2 Casedist

lemma *ex-one-defined-iff*:
$(\exists x.\ P\ x \wedge x \neq \bot) = P\ ONE$
⟨*proof*⟩

lemma *ex-up-defined-iff*:
$(\exists x.\ P\ x \wedge x \neq \bot) = (\exists x.\ P\ (up \cdot x))$
⟨*proof*⟩

lemma *ex-sprod-defined-iff*:
$(\exists y.\ P\ y \wedge y \neq \bot) =$
$(\exists x\ y.\ (P\ (:x,\ y:) \wedge x \neq \bot) \wedge y \neq \bot)$
⟨*proof*⟩

lemma *ex-sprod-up-defined-iff*:
$(\exists y.\ P\ y \wedge y \neq \bot) =$
$(\exists x\ y.\ P\ (:up \cdot x,\ y:) \wedge y \neq \bot)$
⟨*proof*⟩

lemma *ex-ssum-defined-iff*:
$(\exists x.\ P\ x \wedge x \neq \bot) =$
$((\exists x.\ P\ (sinl \cdot x) \wedge x \neq \bot) \vee$

$(\exists\, x.\ P\ (sinr{\cdot}x) \wedge x \neq \bot))$
$\langle proof \rangle$

lemma *exh-start*: $p = \bot \vee (\exists\, x.\ p = x \wedge x \neq \bot)$
$\langle proof \rangle$

lemmas *ex-defined-iffs* =
ex-ssum-defined-iff
ex-sprod-up-defined-iff
ex-sprod-defined-iff
ex-up-defined-iff
ex-one-defined-iff

Rules for turning exh into casedist

lemma *exh-casedist0*: $[\![R;\ R \Longrightarrow P]\!] \Longrightarrow P$
$\langle proof \rangle$

lemma *exh-casedist1*: $((P \vee Q \Longrightarrow R) \Longrightarrow S) \equiv ([\![P \Longrightarrow R;\ Q \Longrightarrow R]\!] \Longrightarrow S)$
$\langle proof \rangle$

lemma *exh-casedist2*: $(\exists\, x.\ P\ x \Longrightarrow Q) \equiv (\bigwedge x.\ P\ x \Longrightarrow Q)$
$\langle proof \rangle$

lemma *exh-casedist3*: $(P \wedge Q \Longrightarrow R) \equiv (P \Longrightarrow Q \Longrightarrow R)$
$\langle proof \rangle$

lemmas *exh-casedists* = *exh-casedist1 exh-casedist2 exh-casedist3*

18.3 Setting up the package

$\langle ML \rangle$

end

theory *HOLCF*
imports *Sprod Ssum Up Lift Discrete One Tr Domain*
uses
holcf-logic.ML
cont-consts.ML
domain/library.ML
domain/syntax.ML
domain/axioms.ML
domain/theorems.ML
domain/extender.ML
domain/interface.ML
adm-tac.ML

begin

$\langle ML \rangle$

end