Mechanically Proving Determinacy of Hierarchical Block Diagram Translations

October 15, 2018

Contents

1 List Operations. Permutations and Substitutions

theory *ListProp* **imports** *Main* ~~/src/HOL/Library/Permutation
begin

lemma *perm-mset*: $perm\ x\ y = (mset\ x = mset\ y)$

lemma *perm-tp*: $perm\ (x@y)\ (y@x)$

lemma *perm-union-left*: $perm\ x\ z \Longrightarrow perm\ (x\ @\ y)\ (z\ @\ y)$

lemma *perm-union-right*: $perm\ x\ z \Longrightarrow perm\ (y\ @\ x)\ (y\ @\ z)$

lemma *perm-trans*: $perm\ x\ y \Longrightarrow perm\ y\ z \Longrightarrow perm\ x\ z$

lemma *perm-sym*: $perm\ x\ y \Longrightarrow perm\ y\ x$

lemma *perm-length*: $perm\ u\ v \Longrightarrow length\ u = length\ v$

lemma *perm-set-eq*: $perm\ x\ y \Longrightarrow set\ x = set\ y$

lemma *perm-empty*[*simp*]: $(perm\ []\ v) = (v = [])$ **and** $(perm\ v\ []) = (v = [])$

lemma *perm-refl*[*simp*]: $perm\ x\ x$

lemma *dist-perm*: $\bigwedge\ y\ .\ distinct\ x \Longrightarrow perm\ x\ y \Longrightarrow distinct\ y$

lemma *split-perm*: $perm\ (a\ \#\ x)\ x' = (\exists\ y\ y'\ .\ x' = y\ @\ a\ \#\ y' \wedge perm\ x\ (y\ @\ y'))$

fun *subst*:: $'a\ list \Rightarrow 'a\ list \Rightarrow 'a \Rightarrow 'a$ **where**
$subst\ []\ []\ c = c\ |$
$subst\ (a\#x)\ (b\#y)\ c = (if\ a = c\ then\ b\ else\ subst\ x\ y\ c)\ |$
$subst\ x\ y\ c = undefined$

lemma *subst-notin* [*simp*]: $\bigwedge\ y\ .\ length\ x = length\ y \Longrightarrow a \notin set\ x \Longrightarrow subst\ x\ y\ a = a$

lemma *subst-cons-a*: $\bigwedge\ y\ .\ distinct\ x \Longrightarrow a \notin set\ x \Longrightarrow b \notin set\ x \Longrightarrow length\ x = length\ y \Longrightarrow subst\ (a\ \#\ x)\ (b\ \#\ y)\ c = (subst\ x\ y\ (subst\ [a]\ [b]\ c))$

lemma *subst-eq*: $subst\ x\ x\ y = y$

fun *Subst* :: $'a\ list \Rightarrow 'a\ list \Rightarrow 'a\ list \Rightarrow 'a\ list$ **where**
$Subst\ x\ y\ [] = []\ |$
$Subst\ x\ y\ (a\ \#\ z) = subst\ x\ y\ a\ \#\ (Subst\ x\ y\ z)$

lemma *Subst-empty*[*simp*]: $Subst\ []\ []\ y = y$

lemma *Subst-eq*: $Subst\ x\ x\ y = y$

lemma *Subst-append*: $Subst\ a\ b\ (x@y) = Subst\ a\ b\ x\ @\ Subst\ a\ b\ y$

lemma *Subst-notin*[*simp*]: $a \notin set\ z \Longrightarrow Subst\ (a\ \#\ x)\ (b\ \#\ y)\ z = Subst\ x\ y\ z$

lemma *Subst-all*[*simp*]: $\bigwedge\ v\ .\ distinct\ u \Longrightarrow length\ u = length\ v \Longrightarrow Subst\ u\ v\ u = v$

lemma *Subst-inex*[*simp*]: $\bigwedge\ b.\ set\ a \cap set\ x = \{\} \Longrightarrow length\ a = length\ b \Longrightarrow Subst\ a\ b\ x = x$

lemma *set-Subst*: $set\ (Subst\ [a]\ [b]\ x) = (if\ a \in set\ x\ then\ (set\ x - \{a\}) \cup \{b\}\ else\ set\ x)$

lemma *distinct-Subst*: $distinct\ (b\#x) \Longrightarrow distinct\ (Subst\ [a]\ [b]\ x)$

lemma *inter-Subst*: $distinct(b\#y) \Longrightarrow set\ x \cap set\ y = \{\} \Longrightarrow b \notin set\ x \Longrightarrow set\ x \cap set\ (Subst\ [a]\ [b]\ y) = \{\}$

lemma *incl-Subst*: $distinct(b\#x) \Longrightarrow set\ y \subseteq set\ x \Longrightarrow set\ (Subst\ [a]\ [b]\ y) \subseteq set\ (Subst\ [a]\ [b]\ x)$

lemma *subst-in-set*: $\bigwedge y.\ length\ x = length\ y \Longrightarrow a \in set\ x \Longrightarrow subst\ x\ y\ a \in set\ y$

lemma *Subst-set-incl*: $length\ x = length\ y \Longrightarrow set\ z \subseteq set\ x \Longrightarrow set\ (Subst\ x\ y\ z) \subseteq set\ y$

lemma *subst-not-in*: $\bigwedge\ y\ .\ a \notin set\ x' \Longrightarrow length\ x = length\ y \Longrightarrow length\ x' = length\ y' \Longrightarrow subst\ (x\ @\ x')\ (y\ @\ y')\ a = subst\ x\ y\ a$

lemma *subst-not-in-b*: $\bigwedge\ y\ .\ a \notin set\ x \Longrightarrow length\ x = length\ y \Longrightarrow length\ x' = length\ y' \Longrightarrow subst\ (x\ @\ x')\ (y\ @\ y')\ a = subst\ x'\ y'\ a$

lemma *Subst-not-in*: $set\ x' \cap set\ z = \{\} \Longrightarrow length\ x = length\ y \Longrightarrow length\ x' = length\ y' \Longrightarrow Subst\ (x\ @\ x')\ (y\ @\ y')\ z = Subst\ x\ y\ z$

lemma *Subst-not-in-a*: $set\ x \cap set\ z = \{\} \Longrightarrow length\ x = length\ y \Longrightarrow length\ x'$

$= length\ y' \Longrightarrow Subst\ (x\ @\ x')\ (y\ @\ y')\ z = Subst\ x'\ y'\ z$

lemma *subst-cancel-right* [*simp*]: $\bigwedge\ y\ z\ .\ set\ x \cap set\ y = \{\} \Longrightarrow length\ y = length\ z \Longrightarrow subst\ (x\ @\ y)\ (x\ @\ z)\ a = subst\ y\ z\ a$

lemma *Subst-cancel-right*: $set\ x \cap set\ y = \{\} \Longrightarrow length\ y = length\ z \Longrightarrow Subst\ (x\ @\ y)\ (x\ @\ z)\ w = Subst\ y\ z\ w$

lemma *subst-cancel-left* [*simp*]: $\bigwedge\ y\ z\ .\ set\ x \cap set\ z = \{\} \Longrightarrow length\ x = length\ y \Longrightarrow subst\ (x\ @\ z)\ (y\ @\ z)\ a = subst\ x\ y\ a$

lemma *Subst-cancel-left*: $set\ x \cap set\ z = \{\} \Longrightarrow length\ x = length\ y \Longrightarrow Subst\ (x\ @\ z)\ (y\ @\ z)\ w = Subst\ x\ y\ w$

lemma *Subst-cancel-right-a*: $a \notin\ set\ y \Longrightarrow length\ y = length\ z \Longrightarrow Subst\ (a\ \#\ y)\ (a\ \#\ z)\ w = Subst\ y\ z\ w$

lemma *subst-subst-id* [*simp*]: $\bigwedge\ y\ .\ a \in set\ y \Longrightarrow distinct\ x \Longrightarrow length\ x = length\ y \Longrightarrow subst\ x\ y\ (subst\ y\ x\ a) = a$

lemma *Subst-Subst-id*[*simp*]: $set\ z \subseteq set\ y \Longrightarrow distinct\ x \Longrightarrow length\ x = length\ y \Longrightarrow Subst\ x\ y\ (Subst\ y\ x\ z) = z$

lemma *Subst-cons-aux-a*: $set\ x \cap set\ y = \{\} \Longrightarrow distinct\ y \Longrightarrow length\ y = length\ z \Longrightarrow Subst\ (x\ @\ y)\ (x\ @\ z)\ y = z$

lemma *Subst-set-empty* [*simp*]: $set\ z \cap set\ x = \{\} \Longrightarrow length\ x = length\ y \Longrightarrow Subst\ x\ y\ z = z$

lemma *length-Subst*[*simp*]: $length\ (Subst\ x\ y\ z) = length\ z$

lemma *subst-Subst*: $\bigwedge\ y\ y'\ .\ length\ y = length\ y' \Longrightarrow a \in set\ w \Longrightarrow subst\ w\ (Subst\ y\ y'\ w)\ a = subst\ y\ y'\ a$

lemma *Subst-Subst*: $length\ y = length\ y' \Longrightarrow set\ z \subseteq set\ w \Longrightarrow Subst\ w\ (Subst\ y\ y'\ w)\ z = Subst\ y\ y'\ z$

primrec *listinter* :: $'a\ list \Rightarrow\ 'a\ list \Rightarrow\ 'a\ list$ (**infixl** $\otimes$ *60*) **where**
$[] \otimes y = []\ |$
$(a\ \#\ x) \otimes y = (if\ a \in set\ y\ then\ a\ \#\ (x \otimes y)\ else\ x \otimes y)$

lemma *inter-filter*: $x \otimes y = filter\ (\lambda\ a\ .\ a \in set\ y)\ x$

lemma *inter-append*: $set\ y \cap set\ z = \{\} \Longrightarrow perm\ (x \otimes (y\ @\ z))\ ((x \otimes y)\ @\ (x \otimes z))$

lemma *append-inter*: $(x @ y) \otimes z = (x \otimes z) @ (y \otimes z)$

lemma *notin-inter* [*simp*]: $a \notin set\ x \Longrightarrow a \notin set\ (x \otimes y)$

lemma *distinct-inter*: $distinct\ x \Longrightarrow distinct\ (x \otimes y)$

lemma *set-inter*: $set\ (x \otimes y) = set\ x \cap set\ y$

primrec *diff* :: $'a\ list \Rightarrow 'a\ list \Rightarrow 'a\ list$ (**infixl** $\ominus$ *52*) **where**
$[] \ominus y = []$ |
$(a \# x) \ominus y = (if\ a \in set\ y\ then\ x \ominus y\ else\ a \# (x \ominus y))$

lemma *diff-filter*: $x \ominus y = filter\ (\lambda\ a\ .\ a \notin set\ y)\ x$

lemma *diff-distinct*: $set\ x \cap set\ y = \{\} \Longrightarrow (y \ominus x) = y$

lemma *set-diff*: $set\ (x \ominus y) = set\ x - set\ y$

lemma *distinct-diff*: $distinct\ x \Longrightarrow distinct\ (x \ominus y)$

definition *addvars* :: $'a\ list \Rightarrow 'a\ list \Rightarrow 'a\ list$ (**infixl** $\oplus$ *55*) **where**
$addvars\ x\ y = x @ (y \ominus x)$

lemma *addvars-distinct*: $set\ x \cap set\ y = \{\} \Longrightarrow x \oplus y = x @ y$

lemma *set-addvars*: $set\ (x \oplus y) = set\ x \cup set\ y$

lemma *distinct-addvars*: $distinct\ x \Longrightarrow distinct\ y \Longrightarrow distinct\ (x \oplus y)$

lemma *mset-inter-diff*: $mset\ oa = mset\ (oa \otimes ia) + mset\ (oa \ominus (oa \otimes ia))$

lemma *diff-inter-left*: $(x \ominus (x \otimes y)) = (x \ominus y)$

lemma *diff-inter-right*: $(x \ominus (y \otimes x)) = (x \ominus y)$

lemma *addvars-minus*: $(x \oplus y) \ominus z = (x \ominus z) \oplus (y \ominus z)$

lemma *addvars-assoc*: $x \oplus y \oplus z = x \oplus (y \oplus z)$

lemma *diff-sym*: $(x \ominus y \ominus z) = (x \ominus z \ominus y)$

lemma *diff-union*: $(x \ominus y @ z) = (x \ominus y \ominus z)$

lemma *diff-notin*: $set\ x \cap set\ z = \{\} \Longrightarrow (x \ominus (y \ominus z)) = (x \ominus y)$

lemma *union-diff*: $x @ y \ominus z = ((x \ominus z) @ (y \ominus z))$

lemma *diff-inter-empty*: $set\ x \cap set\ y = \{\} \Longrightarrow x \ominus y \otimes z = x$

lemma *inter-diff-empty*: $set\ x \cap set\ z = \{\} \Longrightarrow x \otimes (y \ominus z) = (x \otimes y)$

lemma *inter-diff-distrib*: $(x \ominus y) \otimes z = ((x \otimes z) \ominus (y \otimes z))$

lemma *diff-emptyset*: $x \ominus [] = x$

lemma *diff-eq*: $x \ominus x = []$

lemma *diff-subset*: $set\ x \subseteq set\ y \Longrightarrow x \ominus y = []$

lemma *empty-inter*: $set\ x \cap set\ y = \{\} \Longrightarrow x \otimes y = []$

lemma *empty-inter-diff*: $set\ x \cap set\ y = \{\} \Longrightarrow x \otimes (y \ominus z) = []$

lemma *inter-addvars-empty*: $set\ x \cap set\ z = \{\} \Longrightarrow x \otimes y\ @\ z = x \otimes y$

lemma *diff-disjoint*: $set\ x \cap set\ y = \{\} \Longrightarrow x \ominus y = x$

lemma *addvars-empty*[*simp*]: $x \oplus [] = x$

lemma *empty-addvars*[*simp*]: $[] \oplus x = x$

lemma *distrib-diff-addvars*: $x \ominus (y\ @\ z) = ((x \ominus y) \otimes (x \ominus z))$

lemma *inter-subset*: $x \otimes (x \ominus y) = (x \ominus y)$

lemma *diff-cancel*: $x \ominus y \ominus (z \ominus y) = (x \ominus y \ominus z)$

lemma *diff-cancel-set*: $set\ x \cap set\ u = \{\} \Longrightarrow x \ominus y \ominus (z \ominus u) = (x \ominus y \ominus z)$

lemma *inter-subset-l1*: $\bigwedge y.\ distinct\ x \Longrightarrow length\ y = 1 \Longrightarrow set\ y \subseteq set\ x \Longrightarrow x \otimes y = y$

lemma *perm-diff-left-inter*: $perm\ (x \ominus y)\ (((x \ominus y) \otimes z)\ @\ ((x \ominus y) \ominus z))$

lemma *perm-diff-right-inter*: $perm\ (x \ominus y)\ (((x \ominus y) \ominus z)\ @\ ((x \ominus y) \otimes z))$

lemma *perm-switch-aux-a*: $perm\ x\ ((x \ominus y)\ @\ (x \otimes y))$

lemma *perm-switch-aux-b*: $perm\ (x\ @\ (y \ominus x))\ ((x \ominus y)\ @\ (x \otimes y)\ @\ (y \ominus x))$

lemma *perm-switch-aux-c*: $distinct\ x \Longrightarrow distinct\ y \Longrightarrow perm\ ((y \otimes x)\ @\ (y \ominus x))\ y$

lemma *perm-switch-aux-d*: $distinct\ x \Longrightarrow distinct\ y \Longrightarrow perm\ (x \otimes y)\ (y \otimes x)$

lemma *perm-switch-aux-e*: $distinct\ x \Longrightarrow distinct\ y \Longrightarrow perm\ ((x \otimes y)\ @\ (y \ominus x))\ ((y \otimes x)\ @\ (y \ominus x))$

lemma *perm-switch-aux-f*: $distinct\ x \Longrightarrow distinct\ y \Longrightarrow perm\ ((x \otimes y)\ @\ (y \ominus x))\ y$

lemma *perm-switch-aux-h*: $distinct\ x \Longrightarrow distinct\ y \Longrightarrow perm\ ((x \ominus y)\ @\ (x \otimes y)\ @\ (y \ominus x))\ ((x \ominus y)\ @\ y)$

lemma *perm-switch*: $distinct\ x \Longrightarrow distinct\ y \Longrightarrow perm\ (x\ @\ (y \ominus x))\ ((x \ominus y)\ @\ y)$

lemma *perm-aux-a*: $distinct\ x \Longrightarrow distinct\ y \Longrightarrow x \otimes y = x \Longrightarrow perm\ (x\ @\ (y \ominus x))\ y$

lemma *ZZZ-a*: $x \oplus (y \ominus x) = (x \oplus y)$

lemma *ZZZ-b*: $set\ (y \otimes z) \cap set\ x = \{\} \Longrightarrow (x \ominus (y \ominus z) \ominus (z \ominus y)) = (x \ominus y \ominus z)$

lemma *subst-subst*: $\bigwedge\ y\ z\ .\ a \in set\ z \Longrightarrow distinct\ x \Longrightarrow length\ x = length\ y \Longrightarrow length\ z = length\ x$
$\Longrightarrow subst\ x\ y\ (subst\ z\ x\ a) = subst\ z\ y\ a$

lemma *Subst-Subst-a*: $set\ u \subseteq set\ z \Longrightarrow distinct\ x \Longrightarrow length\ x = length\ y \Longrightarrow length\ z = length\ x$
$\Longrightarrow Subst\ x\ y\ (Subst\ z\ x\ u) = (Subst\ z\ y\ u)$

lemma *subst-in*: $\bigwedge\ x'\ .\ length\ x = length\ x' \Longrightarrow a \in set\ x \Longrightarrow subst\ (x\ @\ y)\ (x'\ @\ y')\ a = subst\ x\ x'\ a$

lemma *subst-switch*: $\bigwedge\ x'\ .\ set\ x \cap set\ y = \{\} \Longrightarrow length\ x = length\ x' \Longrightarrow length\ y = length\ y'$
$\Longrightarrow subst\ (x\ @\ y)\ (x'\ @\ y')\ a = subst\ (y\ @\ x)\ (y'\ @\ x')\ a$

lemma *Subst-switch*: $set\ x \cap set\ y = \{\} \Longrightarrow length\ x = length\ x' \Longrightarrow length\ y = length\ y'$
$\Longrightarrow Subst\ (x\ @\ y)\ (x' @\ y')\ z = Subst\ (y\ @\ x)\ (y' @\ x')\ z$

lemma *subst-comp*: $\bigwedge\ x'\ .\ set\ x \cap set\ y = \{\} \Longrightarrow set\ x' \cap set\ y = \{\} \Longrightarrow length\ x = length\ x'$
$\Longrightarrow length\ y = length\ y' \Longrightarrow subst\ (x\ @\ y)\ (x'\ @\ y')\ a = subst\ y\ y'\ (subst\ x\ x'\ a)$

lemma *Subst-comp*: $set\ x \cap set\ y = \{\} \Longrightarrow set\ x' \cap set\ y = \{\} \Longrightarrow length\ x = length\ x'$
$\Longrightarrow length\ y = length\ y' \Longrightarrow Subst\ (x\ @\ y)\ (x'\ @\ y')\ z = Subst\ y\ y'\ (Subst\ x\ x'\ z)$

lemma *set-subst*: $\bigwedge u'$. $\mathit{length}\ u = \mathit{length}\ u' \Longrightarrow \mathit{subst}\ u\ u'\ a \in \mathit{set}\ u' \cup (\{a\} - \mathit{set}\ u)$

lemma *set-Subst-a*: $\mathit{length}\ u = \mathit{length}\ u' \Longrightarrow \mathit{set}\ (\mathit{Subst}\ u\ u'\ z) \subseteq \mathit{set}\ u' \cup (\mathit{set}\ z - \mathit{set}\ u)$

lemma *set-SubstI*: $\mathit{length}\ u = \mathit{length}\ u' \Longrightarrow \mathit{set}\ u' \cup (\mathit{set}\ z - \mathit{set}\ u) \subseteq X \Longrightarrow \mathit{set}\ (\mathit{Subst}\ u\ u'\ z) \subseteq X$

lemma *not-in-set-diff*: $a \notin \mathit{set}\ x \Longrightarrow x \ominus \mathit{ys}\ @\ a\ \#\ \mathit{zs} = x \ominus \mathit{ys}\ @\ \mathit{zs}$

lemma [*simp*]: $(X \cap (Y \cup Z) = \{\}) = (X \cap Y = \{\} \wedge X \cap Z = \{\})$

lemma *Comp-assoc-new-subst-aux*: $\mathit{set}\ u \cap \mathit{set}\ y \cap \mathit{set}\ z = \{\} \Longrightarrow \mathit{distinct}\ z \Longrightarrow \mathit{length}\ u = \mathit{length}\ u'$
$\Longrightarrow \mathit{Subst}\ (z \ominus v)\ (\mathit{Subst}\ u\ u'\ (z \ominus v))\ z = \mathit{Subst}\ (u \ominus y \ominus v)\ (\mathit{Subst}\ u\ u'\ (u \ominus y \ominus v))\ z$

lemma [*simp*]: $(x \ominus y \ominus (y \ominus z)) = (x \ominus y)$

lemma [*simp*]: $(x \ominus y \ominus (y \ominus z \ominus z')) = (x \ominus y)$

lemma *diff-addvars*: $x \ominus (y \oplus z) = (x \ominus y \ominus z)$

lemma *diff-redundant-a*: $x \ominus y \ominus z \ominus (y \ominus u) = (x \ominus y \ominus z)$

lemma *diff-redundant-b*: $x \ominus y \ominus z \ominus (z \ominus u) = (x \ominus y \ominus z)$

lemma *diff-redundant-c*: $x \ominus y \ominus z \ominus (y \ominus u \ominus v) = (x \ominus y \ominus z)$

lemma *diff-redundant-d*: $x \ominus y \ominus z \ominus (z \ominus u \ominus v) = (x \ominus y \ominus z)$

lemma *set-list-empty*: $\mathit{set}\ x = \{\} \Longrightarrow x = []$

lemma [*simp*]: $(x \ominus x \otimes y) \otimes (y \ominus x \otimes y) = []$

lemma [*simp*]: $\mathit{set}\ x \cap \mathit{set}\ (y \ominus x) = \{\}$

lemma [*simp*]: $\mathit{distinct}\ x \Longrightarrow \mathit{distinct}\ y \Longrightarrow \mathit{set}\ x \subseteq \mathit{set}\ y \Longrightarrow \mathit{perm}\ (x\ @\ (y \ominus x))\ y$

lemma [*simp*]: $\mathit{perm}\ x\ y \Longrightarrow \mathit{set}\ x \subseteq \mathit{set}\ y$
lemma [*simp*]: $\mathit{perm}\ x\ y \Longrightarrow \mathit{set}\ y \subseteq \mathit{set}\ x$

lemma [*simp*]: $\mathit{set}\ (x \ominus y) \subseteq \mathit{set}\ x$

lemma *perm-diff* [*simp*]: $\bigwedge x'$. *perm* $x\ x' \Longrightarrow$ *perm* $y\ y' \Longrightarrow$ *perm* $(x \ominus y)$ $(x' \ominus y')$

lemma [*simp*]: *perm* $x\ x' \Longrightarrow$ *perm* $y\ y' \Longrightarrow$ *perm* $(x\ @\ y)\ (x'\ @\ y')$

lemma [*simp*]: *perm* $x\ x' \Longrightarrow$ *perm* $y\ y' \Longrightarrow$ *perm* $(x \oplus y)\ (x' \oplus y')$

thm *distinct-diff*

declare *distinct-diff* [*simp*]

lemma [*simp*]: $\bigwedge x'$. *perm* $x\ x' \Longrightarrow$ *perm* $y\ y' \Longrightarrow$ *perm* $(x \otimes y)\ (x' \otimes y')$

declare *distinct-inter* [*simp*]

lemma *perm-ops*: *perm* $x\ x' \Longrightarrow$ *perm* $y\ y' \Longrightarrow f = (\otimes) \lor f = (\ominus) \lor f = (\oplus)$ $\Longrightarrow$ *perm* $(f\ x\ y)\ (f\ x'\ y')$

lemma [*simp*]: *perm* $x'\ x \Longrightarrow$ *perm* $y'\ y \Longrightarrow f = (\otimes) \lor f = (\ominus) \lor f = (\oplus)$ $\Longrightarrow$ *perm* $(f\ x\ y)\ (f\ x'\ y')$

lemma [*simp*]: *perm* $x\ x' \Longrightarrow$ *perm* $y'\ y \Longrightarrow f = (\otimes) \lor f = (\ominus) \lor f = (\oplus)$ $\Longrightarrow$ *perm* $(f\ x\ y)\ (f\ x'\ y')$

lemma [*simp*]: *perm* $x'\ x \Longrightarrow$ *perm* $y\ y' \Longrightarrow f = (\otimes) \lor f = (\ominus) \lor f = (\oplus)$ $\Longrightarrow$ *perm* $(f\ x\ y)\ (f\ x'\ y')$

lemma *diff-cons*: $(x \ominus (a\ \#\ y)) = (x \ominus [a] \ominus y)$

lemma [*simp*]: $x \oplus y \oplus x = x \oplus y$

lemma *subst-subst-inv*: $\bigwedge y$. *distinct* $y \Longrightarrow$ *length* $x =$ *length* $y \Longrightarrow a \in$ *set* $x \Longrightarrow$ *subst* $y\ x$ (*subst* $x\ y\ a) = a$

lemma *Subst-Subst-inv*: *distinct* $y \Longrightarrow$ *length* $x =$ *length* $y \Longrightarrow$ *set* $z \subseteq$ *set* x $\Longrightarrow$ *Subst* $y\ x$ (*Subst* $x\ y\ z) = z$

lemma *perm-append*: *perm* $x\ x' \Longrightarrow$ *perm* $y\ y' \Longrightarrow$ *perm* $(x\ @\ y)\ (x'\ @\ y')$

lemma $x' = y\ @\ a\ \#\ y' \Longrightarrow$ *perm* $x\ (y\ @\ y') \Longrightarrow$ *perm* $(a\ \#\ x)\ x'$

lemma *perm-diff-eq*: $perm\ y\ y' \Longrightarrow (x \ominus y) = (x \ominus y')$

lemma [*simp*]: $A \cap B = \{\} \Longrightarrow x \in A \Longrightarrow x \in B \Longrightarrow False$
lemma [*simp*]: $A \cap B = \{\} \Longrightarrow x \in A \Longrightarrow x \notin B$

lemma [*simp*]: $B \cap A = \{\} \Longrightarrow x \in A \Longrightarrow x \notin B$
lemma [*simp*]: $B \cap A = \{\} \Longrightarrow x \in A \Longrightarrow x \in B \Longrightarrow False$

lemma *distinct-perm-set-eq*: $distinct\ x \Longrightarrow distinct\ y \Longrightarrow perm\ x\ y = (set\ x = set\ y)$

lemma *set-perm*: $distinct\ x \Longrightarrow distinct\ y \Longrightarrow set\ x = set\ y \Longrightarrow perm\ x\ y$

lemma *distinct-perm-switch*: $distinct\ x \Longrightarrow distinct\ y \Longrightarrow perm\ (x \oplus y)\ (y \oplus x)$

lemma *listinter-diff*: $(x \otimes y) \ominus z = (x \ominus z) \otimes (y \ominus z)$

lemma *set-listinter*: $set\ y = set\ z \Longrightarrow x \otimes y = x \otimes z$

lemma *AAA-c*: $a \notin set\ x \Longrightarrow x \ominus [a] = x$

lemma *distinct-perm-cons*: $distinct\ x \Longrightarrow perm\ (a\ \#\ y)\ x \Longrightarrow perm\ y\ (x \ominus [a])$

lemma *listinter-empty*[*simp*]: $y \otimes [] = []$

lemma *subsetset-inter*: $set\ x \subseteq set\ y \Longrightarrow (x \otimes y) = x$

lemma *addvars-addsame*: $x \oplus y \oplus (x \ominus z) = x \oplus y$

lemma *ZZZ*: $x \ominus x \oplus y = []$

lemma *perm-dist-mem*: $distinct\ x \Longrightarrow a \in set\ x \Longrightarrow perm\ (a\ \#\ (x \ominus [a]))\ x$

lemma *addvars-diff*: $b\ \#\ (x \oplus (z \ominus [b])) = (b\ \#\ x) \oplus z$

lemma *perm-cons*: $a \in set\ y \Longrightarrow distinct\ y \Longrightarrow perm\ x\ (y \ominus [a]) \Longrightarrow perm\ (a\ \#\ x)\ y$

end

2 Translation of Hierarchical Block Diagrams

3 Abstract Algebra of Hierarchical Block Diagrams (except one axiom for feedback)

theory *HBDAlgebra* **imports** *ListProp*
begin

locale *BaseOperationFeedbackless* =

fixes *TI TO* :: $'a \Rightarrow 'tp\ list$

fixes *ID* :: $'tp\ list \Rightarrow 'a$
assumes [*simp*]: $TI(ID\ ts) = ts$
assumes [*simp*]: $TO(ID\ ts) = ts$

fixes *comp* :: $'a \Rightarrow 'a \Rightarrow 'a$ (**infixl** *oo 70*)
assumes *TI-comp*[*simp*]: $TI\ S' = TO\ S \Longrightarrow TI\ (S\ oo\ S') = TI\ S$
assumes *TO-comp*[*simp*]: $TI\ S' = TO\ S \Longrightarrow TO\ (S\ oo\ S') = TO\ S'$
assumes *comp-id-left* [*simp*]: $ID\ (TI\ S)\ oo\ S = S$
assumes *comp-id-right* [*simp*]: $S\ oo\ ID\ (TO\ S) = S$
assumes *comp-assoc*: $TI\ T = TO\ S \Longrightarrow TI\ R = TO\ T \Longrightarrow S\ oo\ T\ oo\ R = S\ oo\ (T\ oo\ R)$

fixes *parallel* :: $'a \Rightarrow 'a \Rightarrow 'a$ (**infixl** $\|$ *80*)
assumes *TI-par* [*simp*]: $TI\ (S \parallel T) = TI\ S\ @\ TI\ T$
assumes *TO-par* [*simp*]: $TO\ (S \parallel T) = TO\ S\ @\ TO\ T$
assumes *par-assoc*: $A \parallel B \parallel C = A \parallel (B \parallel C)$
assumes *empty-par*[*simp*]: $ID\ [] \parallel S = S$
assumes *par-empty*[*simp*]: $S \parallel ID\ [] = S$
assumes *parallel-ID* [*simp*]: $ID\ ts \parallel ID\ ts' = ID\ (ts\ @\ ts')$

assumes *comp-parallel-distrib*: $TO\ S = TI\ S' \Longrightarrow TO\ T = TI\ T' \Longrightarrow (S \parallel T)\ oo\ (S' \parallel T') = (S\ oo\ S') \parallel (T\ oo\ T')$

fixes *Split* :: $'tp\ list \Rightarrow 'a$
fixes *Sink* :: $'tp\ list \Rightarrow 'a$
fixes *Switch* :: $'tp\ list \Rightarrow 'tp\ list \Rightarrow 'a$

assumes *TI-Split*[*simp*]: $TI\ (Split\ ts) = ts$
assumes *TO-Split*[*simp*]: $TO\ (Split\ ts) = ts\ @\ ts$

assumes *TI-Sink*[*simp*]: $TI\ (Sink\ ts) = ts$

assumes *TO-Sink*[*simp*]: $TO\ (Sink\ ts) = []$

assumes *TI-Switch*[*simp*]: $TI\ (Switch\ ts\ ts') = ts\ @\ ts'$
assumes *TO-Switch*[*simp*]: $TO\ (Switch\ ts\ ts') = ts'\ @\ ts$

assumes *Split-Sink-id*[*simp*]: $Split\ ts\ oo\ Sink\ ts \parallel ID\ ts = ID\ ts$

assumes *Split-Switch*[*simp*]: $Split\ ts\ oo\ Switch\ ts\ ts = Split\ ts$
assumes *Split-assoc*: $Split\ ts\ oo\ ID\ ts \parallel Split\ ts = Split\ ts\ oo\ Split\ ts \parallel ID\ ts$

assumes *Switch-append*: $Switch\ ts\ (ts'\ @\ ts'') = Switch\ ts\ ts' \parallel ID\ ts''\ oo\ ID\ ts' \parallel Switch\ ts\ ts''$
assumes *Sink-append*: $Sink\ ts \parallel Sink\ ts' = Sink\ (ts\ @\ ts')$
assumes *Split-append*: $Split\ (ts\ @\ ts') = Split\ ts \parallel Split\ ts'\ oo\ ID\ ts \parallel Switch\ ts\ ts' \parallel ID\ ts'$

assumes *switch-par-no-vars*: $TI\ A = ti \Longrightarrow TO\ A = to \Longrightarrow TI\ B = ti' \Longrightarrow TO\ B = to' \Longrightarrow Switch\ ti\ ti'\ oo\ B \parallel A\ oo\ Switch\ to'\ to = A \parallel B$

fixes $fb :: 'a \Rightarrow 'a$
assumes *TI-fb*: $TI\ S = t\ \#\ ts \Longrightarrow TO\ S = t\ \#\ ts' \Longrightarrow TI\ (fb\ S) = ts$
assumes *TO-fb*: $TI\ S = t\ \#\ ts \Longrightarrow TO\ S = t\ \#\ ts' \Longrightarrow TO\ (fb\ S) = ts'$
assumes *fb-comp*: $TI\ S = t\ \#\ TO\ A \Longrightarrow TO\ S = t\ \#\ TI\ B \Longrightarrow fb\ (ID\ [t] \parallel A\ oo\ S\ oo\ ID\ [t] \parallel B) = A\ oo\ fb\ S\ oo\ B$
assumes *fb-par-indep*: $TI\ S = t\ \#\ ts \Longrightarrow TO\ S = t\ \#\ ts' \Longrightarrow fb\ (S \parallel T) = fb\ S \parallel T$

assumes *fb-switch*: $fb\ (Switch\ [t]\ [t]) = ID\ [t]$

begin
definition $fbtype\ S\ tsa\ ts\ ts' = (TI\ S = tsa\ @\ ts \land TO\ S = tsa\ @\ ts')$

lemma *fb-comp-fbtype*: $fbtype\ S\ [t]\ (TO\ A)\ (TI\ B)$
$\Longrightarrow fb\ ((ID\ [t] \parallel A)\ oo\ S\ oo\ (ID\ [t] \parallel B)) = A\ oo\ fb\ S\ oo\ B$

lemma *fb-serial-no-vars*: $TO\ A = t\ \#\ ts \Longrightarrow TI\ B = t\ \#\ ts$
$\Longrightarrow fb\ (\ ID\ [t] \parallel A\ oo\ Switch\ [t]\ [t] \parallel ID\ ts\ oo\ ID\ [t] \parallel B) = A\ oo\ B$

lemma *TI-fb-fbtype*: $fbtype\ S\ [t]\ ts\ ts' \Longrightarrow TI\ (fb\ S) = ts$

lemma *TO-fb-fbtype*: $fbtype\ S\ [t]\ ts\ ts' \Longrightarrow TO\ (fb\ S) = ts'$

lemma *fb-par-indep-fbtype*: $fbtype\ S\ [t]\ ts\ ts' \Longrightarrow fb\ (S \parallel T) = fb\ S \parallel T$

lemma *comp-id-left-simp* [*simp*]: $TI\ S = ts \Longrightarrow ID\ ts\ oo\ S\ = S$

lemma *comp-id-right-simp* [*simp*]: $TO\ S = ts \Longrightarrow S\ oo\ ID\ ts = S$

lemma *par-Sink-comp*: $TI\ A = TO\ B \Longrightarrow B \parallel Sink\ t\ oo\ A = (B\ oo\ A) \parallel Sink\ t$

lemma *Sink-par-comp*: $TI\ A = TO\ B \Longrightarrow Sink\ t \parallel B\ oo\ A = Sink\ t \parallel (B\ oo\ A)$

lemma *Split-Sink-par*[*simp*]: $TI\ A = ts \Longrightarrow Split\ ts\ oo\ Sink\ ts \parallel A = A$

lemma *Switch-Switch-ID*[*simp*]: $Switch\ ts\ ts'\ oo\ Switch\ ts'\ ts = ID\ (ts\ @\ ts')$

lemma *Switch-parallel*: $TI\ A = ts' \Longrightarrow TI\ B = ts \Longrightarrow Switch\ ts\ ts'\ oo\ A \parallel B = B \parallel A\ oo\ Switch\ (TO\ B)\ (TO\ A)$

lemma *Switch-type-empty*[*simp*]: $Switch\ ts\ [] = ID\ ts$

lemma *Switch-empty-type*[*simp*]: $Switch\ []\ ts = ID\ ts$

lemma *Split-id-Sink*[*simp*]: $Split\ ts\ oo\ ID\ ts \parallel Sink\ ts = ID\ ts$

lemma *Split-par-Sink*[*simp*]: $TI\ A = ts \Longrightarrow Split\ ts\ oo\ A \parallel Sink\ ts = A$

lemma *Split-empty* [*simp*]: $Split\ [] = ID\ []$

lemma *Sink-empty*[*simp*]: $Sink\ [] = ID\ []$

lemma *Switch-Split*: $Switch\ ts\ ts' = Split\ (ts\ @\ ts')\ oo\ Sink\ ts \parallel ID\ ts' \parallel ID\ ts \parallel Sink\ ts'$

lemma *Sink-cons*: $Sink\ (t\ \#\ ts) = Sink\ [t] \parallel Sink\ ts$

lemma *Split-cons*: $Split\ (t\ \#\ ts) = Split\ [t] \parallel Split\ ts\ oo\ ID\ [t] \parallel Switch\ [t]\ ts \parallel ID\ ts$

lemma *Split-assoc-comp*: $TI\ A = ts \Longrightarrow TI\ B = ts \Longrightarrow TI\ C = ts \Longrightarrow Split\ ts\ oo\ A \parallel (Split\ ts\ oo\ B \parallel C) = Split\ ts\ oo\ (Split\ ts\ oo\ A \parallel B) \parallel C$

lemma *Split-Split-Switch*: $Split\ ts\ oo\ Split\ ts \parallel Split\ ts\ oo\ ID\ ts \parallel Switch\ ts\ ts \parallel ID\ ts = Split\ ts\ oo\ Split\ ts \parallel Split\ ts$

lemma *parallel-empty-commute*: $TI\ A = [] \Longrightarrow TO\ B = [] \Longrightarrow A \parallel B = B \parallel A$

lemma *comp-assoc-middle-ext*: $TI\ S2 = TO\ S1 \Longrightarrow TI\ S3 = TO\ S2 \Longrightarrow TI\ S4 = TO\ S3 \Longrightarrow TI\ S5 = TO\ S4 \Longrightarrow$
$S1\ oo\ (S2\ oo\ S3\ oo\ S4)\ oo\ S5 = (S1\ oo\ S2)\ oo\ S3\ oo\ (S4\ oo\ S5)$

lemma *fb-gen-parallel*: $\bigwedge\ S\ .\ fbtype\ S\ tsa\ ts\ ts' \Longrightarrow (fb\ \hat{}\hat{}\ (length\ tsa))\ (S \parallel T) = ((fb\ \hat{}\hat{}\ (length\ tsa))\ (S)) \parallel T$

lemmas *parallel-ID-sym* = *parallel-ID* [*THEN sym*]
declare *parallel-ID* [*simp del*]

lemma *fb-indep*: $\bigwedge S.\ fbtype\ S\ tsa\ (TO\ A)\ (TI\ B) \Longrightarrow (fb\ \hat{}\hat{}\ (length\ tsa))\ ((ID\ tsa \parallel A)\ oo\ S\ oo\ (ID\ tsa \parallel B)) = A\ oo\ (fb\ \hat{}\hat{}\ (length\ tsa))\ S\ oo\ B$

lemma *fb-indep-a*: $\bigwedge S.\ fbtype\ S\ tsa\ (TO\ A)\ (TI\ B) \Longrightarrow length\ tsa = n \Longrightarrow (fb\ \hat{}\hat{}\ n)\ ((ID\ tsa \parallel A)\ oo\ S\ oo\ (ID\ tsa \parallel B)) = A\ oo\ (fb\ \hat{}\hat{}\ n)\ S\ oo\ B$

lemma *fb-comp-right*: $fbtype\ S\ [t]\ ts\ (TI\ B) \Longrightarrow fb\ (S\ oo\ (ID\ [t] \parallel B)) = fb\ S\ oo\ B$

lemma *fb-comp-left*: $fbtype\ S\ [t]\ (TO\ A)\ ts \Longrightarrow fb\ ((ID\ [t] \parallel A)\ oo\ S) = A\ oo\ fb\ S$

lemma *fb-indep-right*: $\bigwedge S.\ fbtype\ S\ tsa\ ts\ (TI\ B) \Longrightarrow (fb\ \hat{}\hat{}\ (length\ tsa))\ (S\ oo\ (ID\ tsa \parallel B)) = (fb\ \hat{}\hat{}\ (length\ tsa))\ S\ oo\ B$

lemma *fb-indep-left*: $\bigwedge S.\ fbtype\ S\ tsa\ (TO\ A)\ ts \Longrightarrow (fb\ \hat{}\hat{}\ (length\ tsa))\ ((ID\ tsa \parallel A)\ oo\ S) = A\ oo\ (fb\ \hat{}\hat{}\ (length\ tsa))\ S$

lemma *TI-fb-fbtype-n*: $\bigwedge S.\ fbtype\ S\ t\ ts\ ts' \Longrightarrow TI\ ((fb\ \hat{}\hat{}\ (length\ t))\ S) = ts$
and *TO-fb-fbtype-n*: $\bigwedge S.\ fbtype\ S\ t\ ts\ ts' \Longrightarrow TO\ ((fb\ \hat{}\hat{}\ (length\ t))\ S) = ts'$

declare *parallel-ID* [*simp*]
end

locale *BaseOperationFeedbacklessVars* = *BaseOperationFeedbackless* +
fixes $TV :: 'var \Rightarrow 'b$
fixes $newvar :: 'var\ list \Rightarrow 'b \Rightarrow 'var$
assumes *newvar-type*[*simp*]: $TV(newvar\ x\ t) = t$
assumes *newvar-distinct* [*simp*]: $newvar\ x\ t \notin set\ x$
assumes $ID\ [TV\ a] = ID\ [TV\ a]$
begin
primrec $TVs :: 'var\ list \Rightarrow 'b\ list$ **where**
$TVs\ [] = []\ |$
$TVs\ (a\ \#\ x) = TV\ a\ \#\ TVs\ x$

lemma *TVs-append*: $TVs\ (x\ @\ y) = TVs\ x\ @\ TVs\ y$

definition $Arb\ t = fb\ (Split\ [t])$

lemma *TI-Arb*[*simp*]: $TI\ (Arb\ t) = []$

lemma *TO-Arb*[*simp*]: $TO\ (Arb\ t) = [t]$

fun *set-var*:: $'var\ list \Rightarrow\ 'var \Rightarrow\ 'a$ **where**
$set\text{-}var\ []\ b = Arb\ (TV\ b)\ |$
$set\text{-}var\ (a\ \#\ x)\ b = (if\ a = b\ then\ ID\ [TV\ a] \parallel Sink\ (TVs\ x)\ else\ Sink\ [TV\ a] \parallel set\text{-}var\ x\ b)$

lemma *TO-set-var*[*simp*]: $TO\ (set\text{-}var\ x\ a) = [TV\ a]$

lemma *TI-set-var*[*simp*]: $TI\ (set\text{-}var\ x\ a) = TVs\ x$

primrec *switch* :: $'var\ list \Rightarrow\ 'var\ list \Rightarrow\ 'a$ ([- $\leadsto$ -]) **where**
$[x \leadsto []] = Sink\ (TVs\ x)\ |$
$[x \leadsto a\ \#\ y] = Split\ (TVs\ x)\ oo\ set\text{-}var\ x\ a \parallel [x \leadsto y]$

lemma *TI-switch*[*simp*]: $TI\ [x \leadsto y] = TVs\ x$

lemma *TO-switch*[*simp*]: $TO\ [x \leadsto y] = TVs\ y$

lemma *switch-not-in-Sink*: $a \notin set\ y \Longrightarrow [a\ \#\ x \leadsto y] = Sink\ [TV\ a] \parallel [x \leadsto y]$

lemma *distinct-id*: $distinct\ x \Longrightarrow [x \leadsto x] = ID\ (TVs\ x)$

lemma *set-var-nin*: $a \notin set\ x \Longrightarrow set\text{-}var\ (x\ @\ y)\ a = Sink\ (TVs\ x) \parallel set\text{-}var\ y\ a$

lemma *set-var-in*: $a \in set\ x \Longrightarrow set\text{-}var\ (x\ @\ y)\ a = set\text{-}var\ x\ a \parallel Sink\ (TVs\ y)$

lemma *set-var-not-in*: $a \notin set\ y \Longrightarrow set\text{-}var\ y\ a = Arb\ (TV\ a) \parallel Sink\ (TVs\ y)$

lemma *set-var-in-a*: $a \notin set\ y \Longrightarrow set\text{-}var\ (x\ @\ y)\ a = set\text{-}var\ x\ a \parallel Sink\ (TVs\ y)$

lemma *switch-append*: $[x \leadsto y\ @\ z] = Split\ (TVs\ x)\ oo\ [x \leadsto y] \parallel [x \leadsto z]$

lemma *switch-nin-a-new*: $set\ x \cap set\ y' = \{\} \Longrightarrow [x\ @\ y \leadsto y'] = Sink\ (TVs\ x) \parallel [y \leadsto y']$

lemma *switch-nin-b-new*: $set\ y \cap set\ z = \{\} \Longrightarrow [x\ @\ y \rightsquigarrow z] = [x \rightsquigarrow z] \parallel Sink\ (TVs\ y)$

lemma *var-switch*: $distinct\ (x\ @\ y) \Longrightarrow [x\ @\ y \rightsquigarrow y\ @\ x] = Switch\ (TVs\ x)\ (TVs\ y)$

lemma *switch-par*: $distinct\ (x\ @\ y) \Longrightarrow distinct\ (u\ @\ v) \Longrightarrow TI\ S = TVs\ x \Longrightarrow TI\ T = TVs\ y \Longrightarrow TO\ S = TVs\ v \Longrightarrow TO\ T = TVs\ u \Longrightarrow$
$S \parallel T = [x\ @\ y \rightsquigarrow y\ @\ x]\ oo\ T \parallel S\ oo\ [u\ @\ v \rightsquigarrow v\ @\ u]$

lemma *par-switch*: $distinct\ (x\ @\ y) \Longrightarrow set\ x' \subseteq set\ x \Longrightarrow set\ y' \subseteq set\ y \Longrightarrow [x \rightsquigarrow x'] \parallel [y \rightsquigarrow y'] = [x\ @\ y \rightsquigarrow x'\ @\ y']$

lemma *set-var-sink*[*simp*]: $a \in set\ x \Longrightarrow (TV\ a) = t \Longrightarrow set\text{-}var\ x\ a\ oo\ Sink\ [t] = Sink\ (TVs\ x)$

lemma *switch-Sink*[*simp*]: $\bigwedge\ ts\ .\ set\ u \subseteq set\ x \Longrightarrow TVs\ u = ts \Longrightarrow [x \rightsquigarrow u]\ oo\ Sink\ ts = Sink\ (TVs\ x)$

lemma *set-var-dup*: $a \in set\ x \Longrightarrow TV\ a = t \Longrightarrow set\text{-}var\ x\ a\ oo\ Split\ [t] = Split\ (TVs\ x)\ oo\ set\text{-}var\ x\ a\ \parallel set\text{-}var\ x\ a$

lemma *switch-dup*: $\bigwedge\ ts\ .\ set\ y \subseteq set\ x \Longrightarrow TVs\ y = ts \Longrightarrow [x \rightsquigarrow y]\ oo\ Split\ ts = Split\ (TVs\ x)\ oo\ [x \rightsquigarrow y] \parallel [x \rightsquigarrow y]$

lemma *TVs-length-eq*: $\bigwedge\ y\ .\ TVs\ x = TVs\ y \Longrightarrow length\ x = length\ y$

lemma *set-var-comp-subst*: $\bigwedge\ y\ .\ set\ u \subseteq set\ x \Longrightarrow TVs\ u = TVs\ y \Longrightarrow a \in set\ y \Longrightarrow [x \rightsquigarrow u]\ oo\ set\text{-}var\ y\ a = set\text{-}var\ x\ (subst\ y\ u\ a)$

lemma *switch-comp-subst*: $set\ u \subseteq set\ x \Longrightarrow set\ v \subseteq set\ y \Longrightarrow TVs\ u = TVs\ y \Longrightarrow [x \rightsquigarrow u]\ oo\ [y \rightsquigarrow v] = [x \rightsquigarrow Subst\ y\ u\ v]$

declare *switch.simps* [*simp del*]

lemma *sw-hd-var*: $distinct\ (a\ \#\ b\ \#\ x) \Longrightarrow [a\ \#\ b\ \#\ x \rightsquigarrow b\ \#\ a\ \#\ x] = Switch\ [TV\ a]\ [TV\ b] \parallel ID\ (TVs\ x)$

lemma *fb-serial*: $distinct\ (a\ \#\ b\ \#\ x) \Longrightarrow TV\ a = TV\ b \Longrightarrow TO\ A = TVs\ (b\ \#\ x) \Longrightarrow TI\ B = TVs\ (a\ \#\ x) \Longrightarrow fb\ ((([[a] \rightsquigarrow [a]] \parallel A)\ oo\ [a\ \#\ b\ \#\ x \rightsquigarrow b\ \#\ a\ \#\ x]\ oo\ ([[b] \rightsquigarrow [b]] \parallel B)) = A\ oo\ B$

lemma *Switch-Split*: $distinct\ x \Longrightarrow [x \rightsquigarrow x\ @\ x] = Split\ (TVs\ x)$

lemma *switch-comp*: $distinct\ x \Longrightarrow perm\ x\ y \Longrightarrow set\ z \subseteq set\ y \Longrightarrow [x \rightsquigarrow y]\ oo$

$[y \leadsto z] = [x \leadsto z]$

lemma *switch-comp-a*: $distinct\ x \Longrightarrow distinct\ y \Longrightarrow set\ y \subseteq set\ x \Longrightarrow set\ z \subseteq set\ y \Longrightarrow [x \leadsto y]\ oo\ [y \leadsto z] = [x \leadsto z]$

primrec $newvars{::}'var\ list \Rightarrow 'b\ list \Rightarrow 'var\ list$ **where**
$newvars\ x\ [] = []\ |$
$newvars\ x\ (t\ \#\ ts) = (let\ y = newvars\ x\ ts\ in\ newvar\ (y@x)\ t\ \#\ y)$

lemma *newvars-type*[*simp*]: $TVs(newvars\ x\ ts) = ts$

lemma *newvars-distinct*[*simp*]: $distinct\ (newvars\ x\ ts)$

lemma *newvars-old-distinct*[*simp*]: $set\ (newvars\ x\ ts) \cap set\ x = \{\}$

lemma *newvars-old-distinct-a*[*simp*]: $set\ x \cap set\ (newvars\ x\ ts) = \{\}$

lemma *newvars-length*: $length(newvars\ x\ ts) = length\ ts$

lemma *TV-subst*[*simp*]: $\bigwedge\ y\ .\ TVs\ x = TVs\ y \Longrightarrow TV\ (subst\ x\ y\ a) = TV\ a$

lemma *TV-Subst*[*simp*]: $TVs\ x = TVs\ y \Longrightarrow TVs\ (Subst\ x\ y\ z) = TVs\ z$

lemma *Subst-cons*: $distinct\ x \Longrightarrow a \notin set\ x \Longrightarrow b \notin set\ x \Longrightarrow length\ x = length\ y$
$\Longrightarrow Subst\ (a\ \#\ x)\ (b\ \#\ y)\ z = Subst\ x\ y\ (Subst\ [a]\ [b]\ z)$

declare *TVs-append* [*simp*]
declare *distinct-id* [*simp*]

lemma *par-empty-right*: $A \parallel [[] \leadsto []] = A$

lemma *par-empty-left*: $[[] \leadsto []] \parallel A = A$
lemma *distinct-vars-comp*: $distinct\ x \Longrightarrow perm\ x\ y \Longrightarrow [x \leadsto y]\ oo\ [y \leadsto x] = ID\ (TVs\ x)$

lemma *comp-switch-id*[*simp*]: $distinct\ x \Longrightarrow TO\ S = TVs\ x \Longrightarrow S\ oo\ [x \leadsto x] = S$

lemma *comp-id-switch*[*simp*]: $distinct\ x \Longrightarrow TI\ S = TVs\ x \Longrightarrow [x \leadsto x]\ oo\ S = S$

lemma *distinct-Subst-a*: $\bigwedge\ v\ .\ a \neq aa \Longrightarrow a \notin set\ v \Longrightarrow aa \notin set\ v \Longrightarrow distinct\ v \Longrightarrow length\ u = length\ v \Longrightarrow subst\ u\ v\ a \neq subst\ u\ v\ aa$

lemma *distinct-Subst-b*: $\bigwedge\ v\ .\ a \notin set\ x \Longrightarrow distinct\ x \Longrightarrow a \notin set\ v \Longrightarrow distinct\ v \Longrightarrow set\ v \cap set\ x = \{\} \Longrightarrow length\ u = length\ v \Longrightarrow subst\ u\ v\ a \notin set\ (Subst\ u\ v\ x)$

lemma *distinct-Subst*: *distinct u* $\Longrightarrow$ *distinct* $(v\ @\ x)$ $\Longrightarrow$ *length u* = *length v* $\Longrightarrow$ *distinct* (*Subst u v x*)

lemma *Subst-switch-more-general*: *distinct u* $\Longrightarrow$ *distinct* $(v\ @\ x)$ $\Longrightarrow$ *set y* $\subseteq$ *set x*
$\Longrightarrow$ *TVs u* = *TVs v* $\Longrightarrow$ $[x \rightsquigarrow y]$ = [*Subst u v x* $\rightsquigarrow$ *Subst u v y*]

lemma *id-par-comp*: *distinct x* $\Longrightarrow$ *TO A* = *TI B* $\Longrightarrow$ $[x \rightsquigarrow x] \parallel (A\ oo\ B)$ = $([x \rightsquigarrow x] \parallel A\ )\ oo\ ([x \rightsquigarrow x] \parallel B)$

lemma *par-id-comp*: *distinct x* $\Longrightarrow$ *TO A* = *TI B* $\Longrightarrow$ $(A\ oo\ B) \parallel [x \rightsquigarrow x]$ = $(A \parallel [x \rightsquigarrow x])\ oo\ (B\ \parallel [x \rightsquigarrow x])$

lemma *switch-parallel-a*: *distinct* $(x\ @\ y)$ $\Longrightarrow$ *distinct* $(u\ @\ v)$ $\Longrightarrow$ *TI S* = *TVs x* $\Longrightarrow$ *TI T* = *TVs y* $\Longrightarrow$ *TO S* = *TVs u* $\Longrightarrow$ *TO T* = *TVs v* $\Longrightarrow$
$S \parallel T\ oo\ [u@v \rightsquigarrow v@u] = [x@y \rightsquigarrow y@x]\ oo\ T \parallel S$

declare *distinct-id* [*simp del*]

lemma *fb-gen-serial*: $\bigwedge A\ B\ v\ x$. *distinct* $(u\ @\ v\ @\ x)$ $\Longrightarrow$ *TO A* = *TVs* $(v@x)$ $\Longrightarrow$ *TI B* = *TVs* $(u\ @\ x)$ $\Longrightarrow$ *TVs u* = *TVs v*
$\Longrightarrow$ (*fb* ^^ *length u*) $(([u \rightsquigarrow u] \parallel A)\ oo\ [u\ @\ v\ @\ x \rightsquigarrow v\ @\ u\ @\ x]\ oo\ ([v \rightsquigarrow v] \parallel B)) = A\ oo\ B$

lemma *fb-par-serial*: *distinct*$(u\ @\ x\ @\ x')$ $\Longrightarrow$ *distinct* $(u\ @\ y\ @\ x')$ $\Longrightarrow$ *TI A* = *TVs x* $\Longrightarrow$ *TO A* = *TVs* $(u@y)$ $\Longrightarrow$ *TI B* = *TVs* $(u@x')$ $\Longrightarrow$ *TO B* = *TVs* y' $\Longrightarrow$
(*fb*^^(*length u*)) $([u\ @\ x\ @\ x' \rightsquigarrow x\ @\ u\ @\ x']\ oo\ (A \parallel B)) = (A \parallel ID\ (TVs\ x')\ oo\ [u\ @\ y\ @\ x' \rightsquigarrow y\ @\ u\ @\ x']\ oo\ ID\ (TVs\ y) \parallel B)$

lemma *switch-newvars*: *distinct x* $\Longrightarrow$ [*newvars w* (*TVs x*) $\rightsquigarrow$ *newvars w* (*TVs x*)] = $[x \rightsquigarrow x]$

lemma *switch-par-comp-Subst*: *distinct x* $\Longrightarrow$ *distinct* y' $\Longrightarrow$ *distinct* z' $\Longrightarrow$ *set y* $\subseteq$ *set x*
$\Longrightarrow$ *set z* $\subseteq$ *set x*
$\Longrightarrow$ *set u* $\subseteq$ *set* y' $\Longrightarrow$ *set v* $\subseteq$ *set* z' $\Longrightarrow$ *TVs y* = *TVs* y' $\Longrightarrow$ *TVs z* = *TVs* z' $\Longrightarrow$
$[x \rightsquigarrow y\ @\ z]\ oo\ [y' \rightsquigarrow u] \parallel [z' \rightsquigarrow v] = [x \rightsquigarrow Subst\ y'\ y\ u\ @\ Subst\ z'\ z\ v]$

lemma *switch-par-comp*: *distinct x* $\Longrightarrow$ *distinct y* $\Longrightarrow$ *distinct z* $\Longrightarrow$ *set y* $\subseteq$

$set\ x \Longrightarrow set\ z \subseteq set\ x$
$\Longrightarrow set\ y' \subseteq set\ y \Longrightarrow set\ z' \subseteq set\ z \Longrightarrow [x \rightsquigarrow y\ @\ z]\ oo\ [y \rightsquigarrow y'] \parallel [z \rightsquigarrow z'] = [x \rightsquigarrow y'\ @\ z']$

lemma *par-switch-eq*: $distinct\ u \Longrightarrow distinct\ v \Longrightarrow distinct\ y' \Longrightarrow distinct\ z'$

$\Longrightarrow TI\ A = TVs\ x \Longrightarrow TO\ A = TVs\ v \Longrightarrow TI\ C = TVs\ v\ @\ TVs\ y \Longrightarrow TVs\ y = TVs\ y' \Longrightarrow$
$TI\ C' = TVs\ v\ @\ TVs\ z \Longrightarrow TVs\ z = TVs\ z' \Longrightarrow$
$set\ x \subseteq set\ u \Longrightarrow set\ y \subseteq set\ u \Longrightarrow set\ z \subseteq set\ u \Longrightarrow$
$[v \rightsquigarrow v] \parallel [u \rightsquigarrow y]\ oo\ C = [v \rightsquigarrow v] \parallel [u \rightsquigarrow z]\ oo\ C'$
$\Longrightarrow [u \rightsquigarrow x\ @\ y]\ oo\ (\ A \parallel [y' \rightsquigarrow y'])\ oo\ C = [u \rightsquigarrow x\ @\ z]\ oo\ (\ A \parallel [z' \rightsquigarrow z'])\ oo\ C'$

lemma *paralle-switch*: $\exists\ x\ y\ u\ v.\ distinct\ (x\ @\ y) \land distinct\ (u\ @\ v) \land TVs\ x = TI\ A$
$\land\ TVs\ u = TO\ A \land TVs\ y = TI\ B \land$
$TVs\ v = TO\ B \land A \parallel B = [x\ @\ y \rightsquigarrow y\ @\ x]\ oo\ (B \parallel A)\ oo\ [v\ @\ u \rightsquigarrow u\ @\ v]$

lemma *par-switch-eq-dist*: $distinct\ (u\ @\ v) \Longrightarrow distinct\ y' \Longrightarrow distinct\ z' \Longrightarrow TI\ A = TVs\ x \Longrightarrow TO\ A = TVs\ v \Longrightarrow TI\ C = TVs\ v\ @\ TVs\ y \Longrightarrow TVs\ y = TVs\ y' \Longrightarrow$
$TI\ C' = TVs\ v\ @\ TVs\ z \Longrightarrow TVs\ z = TVs\ z' \Longrightarrow$
$set\ x \subseteq set\ u \Longrightarrow set\ y \subseteq set\ u \Longrightarrow set\ z \subseteq set\ u \Longrightarrow$
$[v\ @\ u \rightsquigarrow v\ @\ y]\ oo\ C = [v\ @\ u \rightsquigarrow v\ @\ z]\ oo\ C' \Longrightarrow [u \rightsquigarrow x\ @\ y]\ oo\ (\ A \parallel [y' \rightsquigarrow y'])\ oo\ C = [u \rightsquigarrow x\ @\ z]\ oo\ (\ A \parallel [z' \rightsquigarrow z'])\ oo\ C'$

lemma *par-switch-eq-dist-a*: $distinct\ (u\ @\ v) \Longrightarrow TI\ A = TVs\ x \Longrightarrow TO\ A = TVs\ v \Longrightarrow TI\ C = TVs\ v\ @\ TVs\ y \Longrightarrow TVs\ y = ty \Longrightarrow TVs\ z = tz \Longrightarrow$
$TI\ C' = TVs\ v\ @\ TVs\ z \Longrightarrow set\ x \subseteq set\ u \Longrightarrow set\ y \subseteq set\ u \Longrightarrow set\ z \subseteq set\ u \Longrightarrow$
$[v\ @\ u \rightsquigarrow v\ @\ y]\ oo\ C = [v\ @\ u \rightsquigarrow v\ @\ z]\ oo\ C' \Longrightarrow [u \rightsquigarrow x\ @\ y]\ oo\ A \parallel ID\ ty\ oo\ C = [u \rightsquigarrow x\ @\ z]\ oo\ A \parallel ID\ tz\ oo\ C'$

lemma *par-switch-eq-a*: $distinct\ (u\ @\ v) \Longrightarrow distinct\ y' \Longrightarrow distinct\ z' \Longrightarrow distinct\ t' \Longrightarrow distinct\ s'$
$\Longrightarrow TI\ A = TVs\ x \Longrightarrow TO\ A = TVs\ v \Longrightarrow TI\ C = TVs\ t\ @\ TVs\ v\ @\ TVs\ y \Longrightarrow TVs\ y = TVs\ y' \Longrightarrow$
$TI\ C' = TVs\ s\ @\ TVs\ v\ @\ TVs\ z \Longrightarrow TVs\ z = TVs\ z' \Longrightarrow TVs\ t = TVs\ t' \Longrightarrow TVs\ s = TVs\ s' \Longrightarrow$
$set\ t \subseteq set\ u \Longrightarrow set\ x \subseteq set\ u \Longrightarrow set\ y \subseteq set\ u \Longrightarrow set\ s \subseteq set\ u \Longrightarrow set\ z \subseteq set\ u \Longrightarrow$
$[u\ @\ v \rightsquigarrow t\ @\ v\ @\ y]\ oo\ C = [u\ @\ v \rightsquigarrow s\ @\ v\ @\ z]\ oo\ C' \Longrightarrow$
$[u \rightsquigarrow t\ @\ x\ @\ y]\ oo\ ([t' \rightsquigarrow t'] \parallel A \parallel [y' \rightsquigarrow y'])\ oo\ C = [u \rightsquigarrow s\ @\ x\ @\ z]\ oo\ ([s' \rightsquigarrow s'] \parallel A \parallel [z' \rightsquigarrow z'])\ oo\ C'$

lemma *length-TVs*: $length\ (TVs\ x) = length\ x$

lemma *comp-par*: $distinct\ x \Longrightarrow set\ y \subseteq set\ x \Longrightarrow [x \rightsquigarrow x\ @\ x]\ oo\ [x \rightsquigarrow y] \parallel [x \rightsquigarrow y] = [x \rightsquigarrow y\ @\ y]$

lemma *Subst-switch-a*: $distinct\ x \Longrightarrow distinct\ y \Longrightarrow set\ z \subseteq set\ x \Longrightarrow TVs\ x = TVs\ y \Longrightarrow [x \rightsquigarrow z] = [y \rightsquigarrow Subst\ x\ y\ z]$

lemma *change-var-names*: $distinct\ a \Longrightarrow distinct\ b \Longrightarrow TVs\ a = TVs\ b \Longrightarrow [a \rightsquigarrow a\ @\ a] = [b \rightsquigarrow b\ @\ b]$

3.1 Deterministic diagrams

definition $deterministic\ S = (Split\ (TI\ S)\ oo\ S \parallel S = S\ oo\ Split\ (TO\ S))$

lemma *deterministic-split*:
assumes $deterministic\ S$
and $distinct\ (a\#x)$
and $TO\ S = TVs\ (a\ \#\ x)$
shows $S = Split\ (TI\ S)\ oo\ (S\ oo\ [\ a\ \#\ x \rightsquigarrow [a]\ ]) \parallel (S\ oo\ [\ a\ \#\ x \rightsquigarrow x\ ])$

lemma *deterministicE*: $deterministic\ A \Longrightarrow distinct\ x \Longrightarrow distinct\ y \Longrightarrow TI\ A = TVs\ x \Longrightarrow TO\ A = TVs\ y$
$\Longrightarrow [x \rightsquigarrow x\ @\ x]\ oo\ (A \parallel A) = A\ oo\ [y \rightsquigarrow y\ @\ y]$

lemma *deterministicI*: $distinct\ x \Longrightarrow distinct\ y \Longrightarrow TI\ A = TVs\ x \Longrightarrow TO\ A = TVs\ y \Longrightarrow$
$[x \rightsquigarrow x\ @\ x]\ oo\ A \parallel A = A\ oo\ [y \rightsquigarrow y\ @\ y] \Longrightarrow deterministic\ A$

lemma *deterministic-switch*: $distinct\ x \Longrightarrow set\ y \subseteq set\ x \Longrightarrow deterministic\ [x \rightsquigarrow y]$

lemma *deterministic-comp*: $deterministic\ A \Longrightarrow deterministic\ B \Longrightarrow TO\ A = TI\ B \Longrightarrow deterministic\ (A\ oo\ B)$

lemma *deterministic-par*: $deterministic\ A \Longrightarrow deterministic\ B \Longrightarrow deterministic\ (A \parallel B)$

end

end

4 Abstract Algebra of Hierarchical Block Diagrams with All Axioms

theory *ExtendedHBDAlgebra* **imports** *HBDAlgebra*

begin

locale *BaseOperation* = *BaseOperationFeedbackless* +
assumes *fb-twice-switch-no-vars*: $TI\ S = t' \# t \# ts \Longrightarrow TO\ S = t' \# t \# ts'$
$\Longrightarrow (fb\ \hat{}\hat{}\ (2::nat))\ (Switch\ [t]\ [t'] \parallel ID\ ts\ oo\ S\ oo\ Switch\ [t']\ [t] \parallel ID\ ts') = (fb\ \hat{}\hat{}\ (2::\ nat))\ S$

locale *BaseOperationVars* = *BaseOperation* + *BaseOperationFeedbacklessVars*
begin
lemma *fb-twice-switch*: $distinct\ (a \# b \# x) \Longrightarrow distinct\ (a \# b \# y) \Longrightarrow TI\ S = TVs\ (b \# a \# x) \Longrightarrow TO\ S = TVs\ (b \# a \# y)$
$\Longrightarrow (fb\ \hat{}\hat{}\ (2::nat))\ ([a \# b \# x \rightsquigarrow b \# a \# x]\ oo\ S\ oo\ [b \# a \# y \rightsquigarrow a \# b \# y]) = (fb\ \hat{}\hat{}\ (2::\ nat))\ S$

lemma *fb-switch-a*: $\bigwedge S\ .\ distinct\ (a \# z\ @\ x) \Longrightarrow distinct\ (a \# z\ @\ y) \Longrightarrow TI\ S = TVs\ (z\ @\ a \# x) \Longrightarrow TO\ S = TVs\ (z\ @\ a \# y)$
$\Longrightarrow (fb\ \hat{}\hat{}\ (Suc\ (length\ z)))\ ([a \# z\ @\ x \rightsquigarrow z\ @\ a \# x]\ oo\ S\ oo\ [z\ @\ a \# y \rightsquigarrow a \# z\ @\ y]) = (fb\ \hat{}\hat{}\ (Suc\ (length\ z)))\ S$

lemma *swap-power*: $(f\ \hat{}\hat{}\ n)\ ((f\ \hat{}\hat{}\ m)\ S) = (f\ \hat{}\hat{}\ m)\ ((f\ \hat{}\hat{}\ n)\ S)$

lemma *fb-switch-b*: $\bigwedge v\ x\ y\ S\ .\ distinct\ (u\ @\ v\ @\ x) \Longrightarrow distinct\ (u\ @\ v\ @\ y) \Longrightarrow TI\ S = TVs\ (v\ @\ u\ @\ x) \Longrightarrow TO\ S = TVs\ (v\ @\ u\ @\ y)$
$\Longrightarrow (fb\ \hat{}\hat{}\ (length\ (u\ @\ v)))\ ([u\ @\ v\ @\ x \rightsquigarrow v\ @\ u\ @\ x]\ oo\ S\ oo\ [v\ @\ u\ @\ y \rightsquigarrow u\ @\ v\ @\ y]) = (fb\ \hat{}\hat{}\ (length\ (u\ @\ v)))\ S$

theorem *fb-perm*: $\bigwedge v\ S\ .\ perm\ u\ v \Longrightarrow distinct\ (u\ @\ x) \Longrightarrow distinct\ (u\ @\ y) \Longrightarrow fbtype\ S\ (TVs\ u)\ (TVs\ x)\ (TVs\ y)$
$\Longrightarrow (fb\ \hat{}\hat{}\ (length\ u))\ ([v\ @\ x \rightsquigarrow u\ @\ x]\ oo\ S\ oo\ [u\ @\ y \rightsquigarrow v\ @\ y]) = (fb\ \hat{}\hat{}\ (length\ u))\ S$
end
end

5 Constructive Functions

theory *Constructive* **imports** *Main*
begin

notation
bot ($\bot$) **and**
top ($\top$) **and**
inf (**infixl** $\sqcap$ *70*)
and *sup* (**infixl** $\sqcup$ *65*)

class *order-bot-max* = *order-bot* +
fixes *maximal* :: $'a \Rightarrow bool$
assumes *maximal-def*: $maximal\ x = (\forall\ y\ .\ \neg\ x < y)$
assumes [*simp*]: $\neg\ maximal\ \bot$

begin
lemma *ex-not-le-bot*[*simp*]: $\exists\ a.\ \neg\ a \leq \bot$
end

instantiation *option* :: (*type*) *order-bot-max*
begin
definition *bot-option-def*: $(\bot{::}'a\ option) = None$
definition *le-option-def*: $((x{::}'a\ option) \leq y) = (x = None \vee x = y)$
definition *less-option-def*: $((x{::}'a\ option) < y) = (x \leq y \wedge \neg\ (y \leq x))$
definition *maximal-option-def*: $maximal\ (x{::}'a\ option) = (\forall\ y\ .\ \neg\ x < y)$

instance

lemma [*simp*]: $None \leq x$
end

context *order-bot*
begin
definition *is-lfp* $f\ x = ((f\ x = x) \wedge (\forall\ y\ .\ f\ y = y \longrightarrow x \leq y))$
definition *emono* $f = (\forall\ x\ y.\ x \leq y \longrightarrow f\ x \leq f\ y)$

definition *Lfp* $f = Eps\ (is\text{-}lfp\ f)$

lemma *lfp-unique*: *is-lfp* $f\ x \Longrightarrow$ *is-lfp* $f\ y \Longrightarrow x = y$

lemma *lfp-exists*: *is-lfp* $f\ x \Longrightarrow Lfp\ f = x$

lemma *emono-a*: *emono* $f \Longrightarrow x \leq y \Longrightarrow f\ x \leq f\ y$

lemma *emono-fix*: *emono* $f \Longrightarrow f\ y = y \Longrightarrow (f\ \hat{}\hat{}\ n)\ \bot \leq y$

lemma *emono-is-lfp*: *emono* $(f{::}'a \Rightarrow 'a) \Longrightarrow (f\ \hat{}\hat{}\ (n + 1))\ \bot = (f\ \hat{}\hat{}\ n)\ \bot \Longrightarrow is\text{-}lfp\ f\ ((f\ \hat{}\hat{}\ n)\ \bot)$

lemma *emono-lfp-bot*: *emono* $(f{::}'a \Rightarrow 'a) \Longrightarrow (f\ \hat{}\hat{}\ (n + 1))\ \bot = (f\ \hat{}\hat{}\ n)\ \bot \Longrightarrow Lfp\ f = ((f\ \hat{}\hat{}\ n)\ \bot)$

lemma *emono-up*: *emono* $f \Longrightarrow (f\ \hat{}\hat{}\ n)\ \bot \leq (f\ \hat{}\hat{}\ (Suc\ n))\ \bot$
end

context *order*
begin
definition *min-set* $A = (SOME\ n\ .\ n \in A \wedge (\forall\ x \in A\ .\ n \leq x))$
end

lemma *min-nonempty-nat-set-aux*: $\forall\ A\ .\ (n{::}nat) \in A \longrightarrow (\exists\ k \in A\ .\ (\forall\ x \in A\ .\ k \leq x))$

lemma *min-nonempty-nat-set*: $(n{::}nat) \in A \Longrightarrow (\exists\ k\ .\ k \in A \wedge (\forall\ x \in A\ .\ k \leq x))$

thm *someI-ex*

lemma *min-set-nat-aux*: $(n{::}nat) \in A \Longrightarrow \mathit{min\text{-}set}\ A \in A \wedge (\forall\ x \in A\ .\ \mathit{min\text{-}set}\ A \leq x)$

lemma $(n{::}nat) \in A \Longrightarrow \mathit{min\text{-}set}\ A \in A \wedge \mathit{min\text{-}set}\ A \leq n$

lemma *min-set-in*: $(n{::}nat) \in A \Longrightarrow \mathit{min\text{-}set}\ A \in A$

lemma *min-set-less*: $(n{::}nat) \in A \Longrightarrow \mathit{min\text{-}set}\ A \leq n$

definition *mono-a* $f = (\forall\ a\ b\ a'\ b'.\ (a{::}'a{::}order) \leq a' \wedge (b{::}'b{::}order) \leq b' \longrightarrow f\ a\ b \leq f\ a'\ b')$

class *fin-cpo* $=$ *order-bot-max* $+$

assumes *fin-up-chain*: $(\forall\ i{::}\ nat\ .\ a\ i \leq a\ (Suc\ i)) \Longrightarrow \exists\ n\ .\ \forall\ i \geq n\ .\ a\ i = a\ n$

begin

lemma *emono-ex-lfp*: $\mathit{emono}\ f \Longrightarrow \exists\ n\ .\ \mathit{is\text{-}lfp}\ f\ ((f\ \hat{}\hat{}\ n)\ \bot)$

lemma *emono-lfp*: $\mathit{emono}\ f \Longrightarrow \exists\ n\ .\ \mathit{Lfp}\ f = (f\ \hat{}\hat{}\ n)\ \bot$

lemma *emono-is-lfp*: $\mathit{emono}\ f \Longrightarrow \mathit{is\text{-}lfp}\ f\ (\mathit{Lfp}\ f)$

definition *lfp-index* $(f{::}'a \Rightarrow 'a) = \mathit{min\text{-}set}\ \{n\ .\ (f\ \hat{}\hat{}\ n)\ \bot = (f\ \hat{}\hat{}\ (n + 1))\ \bot\}$

lemma *lfp-index-aux*: $\mathit{emono}\ f \Longrightarrow (\forall\ i < (\mathit{lfp\text{-}index}\ f)\ .\ (f\ \hat{}\hat{}\ i)\ \bot < (f\ \hat{}\hat{}\ (i + 1))\ \bot) \wedge (f\ \hat{}\hat{}\ (\mathit{lfp\text{-}index}\ f))\ \bot = (f\ \hat{}\hat{}\ ((\mathit{lfp\text{-}index}\ f) + 1))\ \bot$

lemma [*simp*]: $\mathit{emono}\ f \Longrightarrow i < \mathit{lfp\text{-}index}\ f \Longrightarrow (f\ \hat{}\hat{}\ i)\ \bot < f\ ((f\ \hat{}\hat{}\ i)\ \bot)$

lemma [*simp*]: $\mathit{emono}\ f \Longrightarrow f\ ((f\ \hat{}\hat{}\ (\mathit{lfp\text{-}index}\ f))\ \bot) = (f\ \hat{}\hat{}\ (\mathit{lfp\text{-}index}\ f))\ \bot$

lemma $\mathit{emono}\ f \Longrightarrow \mathit{Lfp}\ f = (f\ \hat{}\hat{}\ \mathit{lfp\text{-}index}\ f)\ \bot$

lemma *AA-aux*: $\mathit{emono}\ f \Longrightarrow (\bigwedge\ b\ .\ b \leq a \Longrightarrow f\ b \leq a) \Longrightarrow (f\ \hat{}\hat{}\ n)\ \bot \leq a$

lemma *AA*: $\mathit{emono}\ f \Longrightarrow (\bigwedge\ b\ .\ b \leq a \Longrightarrow f\ b \leq a) \Longrightarrow \mathit{Lfp}\ f \leq a$

lemma *BB*: $\mathit{emono}\ f \Longrightarrow f\ (\mathit{Lfp}\ f) = \mathit{Lfp}\ f$

lemma *Lfp-mono*: $emono\ f \Longrightarrow emono\ g \Longrightarrow (\bigwedge\ a\ .\ f\ a \leq g\ a) \Longrightarrow Lfp\ f \leq Lfp\ g$

end
declare [[*show-types*]]

lemma [*simp*]: $mono\text{-}a\ f \Longrightarrow emono\ (f\ a)$

lemma [*simp*]: $mono\text{-}a\ f \Longrightarrow emono\ (\lambda\ a\ .\ f\ a\ b)$

lemma *mono-aD*: $mono\text{-}a\ f \Longrightarrow a \leq a' \Longrightarrow b \leq b' \Longrightarrow f\ a\ b \leq f\ a'\ b'$

lemma [*simp*]: $mono\text{-}a\ (f::'a::fin\text{-}cpo \Rightarrow 'b::fin\text{-}cpo \Rightarrow 'b) \Longrightarrow mono\text{-}a\ g \Longrightarrow emono\ (\lambda b.\ f\ (Lfp\ (g\ b))\ b)$

lemma *CCC*: $mono\text{-}a\ (f::'a::fin\text{-}cpo \Rightarrow 'b::fin\text{-}cpo \Rightarrow 'b) \Longrightarrow mono\text{-}a\ g \Longrightarrow Lfp\ (\lambda a.\ g\ (Lfp\ (f\ a))\ a) \leq Lfp\ (g\ (Lfp\ (\lambda b.\ f\ (Lfp\ (g\ b))\ b)))$

lemma *Lfp-commute*: $mono\text{-}a\ (f::'a::fin\text{-}cpo \Rightarrow 'b::fin\text{-}cpo \Rightarrow 'b::fin\text{-}cpo) \Longrightarrow mono\text{-}a\ g \Longrightarrow Lfp\ (\lambda\ b\ .\ f\ (Lfp\ (\lambda\ a\ .\ (g\ (Lfp\ (f\ a)))\ a))\ b) = Lfp\ (\lambda\ b\ .\ f\ (Lfp\ (g\ b))\ b)$

instantiation *option* :: (*type*) *fin-cpo*
begin
lemma *fin-up-non-bot*: $(\forall\ i\ .\ (a::nat \Rightarrow 'a\ option)\ i \leq a\ (Suc\ i)) \Longrightarrow a\ n \neq \bot \Longrightarrow n \leq i \Longrightarrow a\ i = a\ n$

lemma *fin-up-chain-option*: $(\forall\ i::\ nat\ .\ (a::nat \Rightarrow 'a\ option)\ i \leq a\ (Suc\ i)) \Longrightarrow \exists\ n\ .\ \forall\ i \geq n\ .\ a\ i = a\ n$

instance
end

instantiation *prod* :: (*order-bot-max*, *order-bot-max*) *order-bot-max*
begin
definition *bot-prod-def*: $(\bot :: 'a \times 'b) = (\bot, \bot)$
definition *le-prod-def*: $(x \leq y) = (fst\ x \leq fst\ y \wedge snd\ x \leq snd\ y)$
definition *less-prod-def*: $((x::'a \times 'b) < y) = (x \leq y \wedge \neg\ (y \leq x))$
definition *maximal-prod-def*: $maximal\ (x::'a \times 'b) = (\forall\ y\ .\ \neg\ x < y)$

instance
end

instantiation *prod* :: (*fin-cpo*, *fin-cpo*) *fin-cpo*
begin

lemma *fin-up-chain-prod*: $(\forall\ i{::}\ nat\ .\ (a{::}nat \Rightarrow 'a \times 'b)\ i \leq a\ (Suc\ i)) \Longrightarrow \exists\ n\ .\ \forall\ i \geq n\ .\ a\ i = a\ n$
instance
end

end

6 Constructive Functions are a Model of the HBD Algebra

theory *ConsFuncHBDModel* **imports** *ExtendedHBDAlgebra Constructive*
begin

datatype *Types* = *int* | *bool* | *nat*

datatype *Values* = *Inte* (*integer* : *int option*) | *Bool* (*boolean*: *bool option*) | *Nat* (*natural*: *nat option*)

primrec *tv* :: *Values* ⇒ *Types* **where**
tv (*Inte i*) = *int* |
tv (*Bool b*) = *bool* |
tv (*Nat n*) = *nat*

primrec *tp* :: *Values list* ⇒ *Types list* **where**
tp [] = [] |
tp (*a* # *v*) = *tv a* # *tp v*

fun *le-val* :: *Values* ⇒ *Values* ⇒ *bool* **where**
(*le-val* (*Inte v*) (*Inte u*)) = ($v \leq u$) |
(*le-val* (*Bool v*) (*Bool u*)) = ($v \leq u$) |
(*le-val* (*Nat v*) (*Nat u*)) = ($v \leq u$) |
le-val - - = *False*

instantiation *Values* :: *order*
begin
definition *le-Values-def*: ((*v*::*Values*) $\leq$ *u*) = *le-val v u*
definition *less-Values-def*: ((*v*::*Values*) < *u*) = ($v \leq u \land \neg\ u \leq v$)
instance
end

fun *le-list* :: *'a*::*order list* ⇒ *'a*::*order list* ⇒ *bool* **where**
le-list [] [] = *True* |
le-list (*a* # *x*) (*b* # *y*) = ($a \leq b \land$ *le-list x y*) |
le-list - - = *False*

instantiation *list* :: (*order*) *order*
begin
definition *le-list-def*: ((*v*::*'a list*) $\leq$ *u*) = *le-list u v*

definition *less-list-def*: $((v::'a\ list) < u) = (v \leq u \wedge \neg\ u \leq v)$
instance
end

lemma [*simp*]: *mono integer*

lemma [*simp*]: *mono boolean*

lemma [*simp*]: *mono natural*

definition *has-in-type* $x = \{f\ .\ (dom\ f = \{v\ .\ tp\ v = x\})\}$
definition *has-out-type* $x = \{f\ .\ (image\ f\ (dom\ f) \subseteq Some\ `\ \{v\ .\ tp\ v = x\})\}$

definition *has-in-out-type* $x\ y$ = *has-in-type* $x \cap$ *has-out-type* y

definition *ID-f* $x\ v = (if\ tp\ v = x\ then\ Some\ v\ else\ None)$

lemma [*simp*]: $(tp\ x = []) = (x = [])$

lemma *map-comp-type*: $f \in$ *has-in-out-type* $x\ y \Longrightarrow g \in$ *has-in-out-type* $y\ z \Longrightarrow$ $g \circ_m f \in$ *has-in-out-type* $x\ z$

definition *TI-f* $f = (SOME\ x\ .\ (\exists\ y\ .\ f \in$ *has-in-out-type* $x\ y))$

definition *TO-f* $f = (SOME\ y\ .\ (\exists\ x\ .\ f \in$ *has-in-out-type* $x\ y))$

fun *pref* :: *Values list* $\Rightarrow$ *Types list* $\Rightarrow$ *Values list* **where**
pref $v\ [] = []\ |$
pref $(a\ \#\ v)\ (t\ \#\ x) = (if\ tv\ a = t\ then\ a\ \#\ pref\ v\ x\ else\ undefined)\ |$
pref $v\ x = undefined$

fun *suff* :: *Values list* $\Rightarrow$ *Types list* $\Rightarrow$ *Values list* **where**
suff $v\ [] = v\ |$
suff $(a\ \#\ v)\ (t\ \#\ x) = (if\ tv\ a = t\ then\ suff\ v\ x\ else\ undefined)\ |$
suff $v\ x = undefined$

lemma *tp-pref-suff*: $\bigwedge x\ y\ .\ tp\ v = x\ @\ y \Longrightarrow tp\ (pref\ v\ x) = x \wedge tp\ (suff\ v\ x) = y$

definition *par-f* $f\ g\ v = (if\ tp\ v = (TI\text{-}f\ f)\ @\ (TI\text{-}f\ g)\ then\ Some\ (the\ (f\ (pref\ v\ (TI\text{-}f\ f)))\ @\ (the\ (g\ (suff\ v\ (TI\text{-}f\ f))))\ else\ None)$

fun *some-v*:: *Types list* $\Rightarrow$ *Values list* **where**
some-v $[] = []\ |$
some-v $(int\ \#\ x) = (Inte\ undefined)\ \#$ *some-v* $x\ |$
some-v $(bool\ \#\ x) = (Bool\ undefined)\ \#$ *some-v* $x\ |$
some-v $(nat\ \#\ x) = (Nat\ undefined)\ \#$ *some-v* x

lemma [*simp*]: $tp\ (some\text{-}v\ x) = x$

lemma *same-in-type*: $f \in has\text{-}in\text{-}type\ x \Longrightarrow f \in has\text{-}in\text{-}type\ y \Longrightarrow x = y$

lemma *same-out-type*: $f \in has\text{-}in\text{-}type\ z \Longrightarrow f \in has\text{-}out\text{-}type\ x \Longrightarrow f \in has\text{-}out\text{-}type\ y \Longrightarrow x = y$

lemma *type-has-type*:
assumes *A*: $f \in has\text{-}in\text{-}out\text{-}type\ x\ y$
shows $TI\text{-}f\ f = x$ **and** $TO\text{-}f\ f = y$

lemma *has-type-out-type*: $f \in has\text{-}in\text{-}out\text{-}type\ x\ y \Longrightarrow tp\ v = x \Longrightarrow tp\ (the\ (f\ v)) = y$

lemma *tp-append*: $tp\ (v\ @\ u) = tp\ v\ @\ tp\ u$

lemma *par-f-type*: $f \in has\text{-}in\text{-}out\text{-}type\ x\ y \Longrightarrow g \in has\text{-}in\text{-}out\text{-}type\ x'\ y' \Longrightarrow par\text{-}f\ f\ g \in has\text{-}in\text{-}out\text{-}type\ (x\ @\ x')\ (y\ @\ y')$

definition $Dup\text{-}f\ x\ v = (if\ tp\ v = x\ then\ Some\ (v\ @\ v)\ else\ None)$

lemma *Dup-has-in-out-type*: $Dup\text{-}f\ x \in has\text{-}in\text{-}out\text{-}type\ x\ (x\ @\ x)$

definition $Sink\text{-}f\ x\ v = (if\ tp\ v = x\ then\ Some\ []\ else\ None)$

lemma *Sink-has-in-out-type*: $Sink\text{-}f\ x \in has\text{-}in\text{-}out\text{-}type\ x\ []$

definition $Switch\text{-}f\ x\ y\ v = (if\ tp\ v = x\ @\ y\ then\ Some\ (suff\ v\ x\ @\ pref\ v\ x)\ else\ None)$

lemma *Switch-has-in-out-type*: $Switch\text{-}f\ x\ y \in has\text{-}in\text{-}out\text{-}type\ (x\ @\ y)\ (y\ @\ x)$

primrec $fb\text{-}t :: Types \Rightarrow (Values \Rightarrow Values) \Rightarrow Values$ **where**
$fb\text{-}t\ int\ f\ = Inte\ (Lfp\ (\lambda\ a\ .\ integer\ (f\ (Inte\ a))))\ |$
$fb\text{-}t\ bool\ f = Bool\ (Lfp\ (\lambda\ a\ .\ boolean\ (f\ (Bool\ a))))\ |$
$fb\text{-}t\ nat\ f\ = Nat\ \ (Lfp\ (\lambda\ a\ .\ natural\ (f\ (Nat\ a))))$

definition $fb\text{-}f\ f\ v = \ (if\ tp\ v = tl\ (TI\text{-}f\ f)\ then\ Some\ (tl\ (the\ (f\ ((fb\text{-}t\ (hd\ (TI\text{-}f\ f))\ (\lambda\ a\ .\ hd\ (the\ (f\ (a\ \#\ v)))))\ \#\ v))))\ else\ None)$

thm *le-Values-def*
thm *le-val.simps*

lemma [*simp*]: *mono Inte*

lemma [*simp*]: *mono Bool*

lemma [*simp*]: *mono Nat*

thm *monoE*
thm *monoI*
thm *mono-aD*
lemma [*simp*]: *mono A* $\Longrightarrow$ *mono B* $\Longrightarrow$ *mono C* $\Longrightarrow$ *mono-a f* $\Longrightarrow$ *mono-a* (λa b. C (f (A a) (B b)))

lemma *fb-t-commute*: *mono-a f* $\Longrightarrow$ *mono-a g*
$\Longrightarrow$ *fb-t t* (λ b . f (*fb-t* t' (λ a . (g (*fb-t t* (f a))) a)) b) = *fb-t t* (λ b . f (*fb-t* t' (g b)) b)

lemma *fb-t-eq-type*: ($\bigwedge$ a . *tv* a = t $\Longrightarrow$ f a = g a) $\Longrightarrow$ *fb-t t f* = *fb-t t g*

lemma [*simp*]: *tv* (*fb-t t f*) = t

lemma *has-type-type-in*: f v = *Some u* $\Longrightarrow$ f $\in$ *has-in-out-type x y* $\Longrightarrow$ *tp v* = x

lemma *has-type-type-in-a*: f v = *None* $\Longrightarrow$ f $\in$ *has-in-out-type x y* $\Longrightarrow$ *tp v* $\neq$ x

lemma *has-type-defined*: f $\in$ *has-in-out-type x y* $\Longrightarrow$ *tp v* = x $\Longrightarrow$ $\exists$ u . f v = *Some u*

lemma *tp-tail*: *tp* (*tl x*) = *tl* (*tp x*)

lemma *fb-type*: f $\in$ *has-in-out-type* (t # x) (t # y) $\Longrightarrow$ *fb-f f* $\in$ *has-in-out-type x y*

lemma [*simp*]: *TI-f* (*Switch-f x y*) = x @ y

lemma *ID-f-type*[*simp*]: *ID-f ts* $\in$ *has-in-out-type ts ts*

lemma [*simp*]: *TI-f* (*ID-f ts*) = *ts*

lemma [*simp*]: *tp v* = *ts* $\Longrightarrow$ *ID-f ts v* = *Some v*

lemma *fb-switch-aux*: f $\in$ *has-in-out-type* (t' # t # *ts*) (t' # t # *ts*$'$) $\Longrightarrow$
par-f (*Switch-f* [t'] [t]) (*ID-f ts*$'$) $\circ_m$ (f $\circ_m$ *par-f* (*Switch-f* [t] [t']) (*ID-f ts*)) =
(λ v . (*if tp v* = t # t' # *ts then case v of a* # b # v' $\Rightarrow$ (*case f* (b # a # v') *of Some* (c # d # u) $\Rightarrow$ *Some* (d # c # u)) *else None*))

lemma *TI-f-fb-f* [*simp*]: $f \in$ *has-in-out-type* $(t\ \#\ ts)$ $(\ t\ \#\ ts')$ $\Longrightarrow$ *TI-f* (*fb-f* f) $= ts$

declare [[*show-types=false*]]

lemma *fb-t-type*: *fb-t* t ($\lambda a.$ *if tv* $a = t$ *then* $f\ a$ *else* $g\ a$) $=$ *fb-t* $t\ f$

lemma *le-values-same-type*: $a \le b \Longrightarrow$ *tv* $a =$ *tv* b

thm *has-type-out-type*

definition *mono-f* $= \{f\ .\ (\forall\ x\ y\ .$ *le-list* $x\ y \longrightarrow$ *le-list* (*the* ($f\ x$)) (*the* ($f\ y$)))$\}$

lemma [*simp*]: *le-list* $v\ v$

lemma *le-pref*: $\bigwedge\ v\ x\ .$ *le-list* $u\ v \Longrightarrow$ *le-list* (*pref* $u\ x$) (*pref* $v\ x$)

lemma *le-suff*: $\bigwedge\ v\ x\ .$ *le-list* $u\ v \Longrightarrow$ *le-list* (*suff* $u\ x$) (*suff* $v\ x$)

lemma *le-list-append*: $\bigwedge\ y\ .$ *le-list* $x\ y \Longrightarrow$ *le-list* $x'\ y' \Longrightarrow$ *le-list* (x @ x') (y @ y')

thm *monoD*

lemma *mono-fD*: $f \in$ *mono-f* $\Longrightarrow$ *le-list* $x\ y \Longrightarrow$ *le-list* (*the* ($f\ x$)) (*the* ($f\ y$))

lemma *le-values-list-same-type*: $\bigwedge$ (y::*Values list*) . *le-list* $x\ y \Longrightarrow$ *tp* $x =$ *tp* y

lemma *map-comp-mono*: $f \in$ *mono-f* $\Longrightarrow g \in$ *mono-f* $\Longrightarrow$ ($\bigwedge\ x\ y\ .$ *tp* $x =$ *tp* $y \Longrightarrow f\ x =$ *None* $\Longrightarrow f\ y =$ *None*) $\Longrightarrow$ ($\bigwedge\ x\ y\ .$ *tp* $x =$ *tp* $y \Longrightarrow g\ x =$ *None* $\Longrightarrow g\ y =$ *None*) $\Longrightarrow g \circ_m f \in$ *mono-f*

lemma *par-mono*: $f \in$ *mono-f* $\Longrightarrow g \in$ *mono-f* $\Longrightarrow$ ($\bigwedge\ x\ y\ .$ *tp* $x\ =$ *tp* $y \Longrightarrow f\ x =$ *None* $\Longrightarrow f\ y =$ *None*) $\Longrightarrow$ ($\bigwedge\ x\ y\ .$ *tp* $x\ =$ *tp* $y \Longrightarrow g\ x =$ *None* $\Longrightarrow g\ y =$ *None*) $\Longrightarrow$ *par-f* $f\ g \in$ *mono-f*

lemma *mono-f-emono*: $f \in$ *mono-f* $\Longrightarrow$ ($\bigwedge\ x\ y\ .$ *tp* $x\ =$ *tp* $y \Longrightarrow f\ x =$ *None* $\Longrightarrow f\ y =$ *None*) $\Longrightarrow$ *mono* $A \Longrightarrow$ *mono* $B \Longrightarrow$ *emono* ($\lambda a.$ A (*hd* (*the* (f ($B\ a$ $\#\ x$)))))

lemma *mono-fb-t-aux*: $f \in$ *mono-f* $\Longrightarrow$
le-list $x\ y \Longrightarrow$ ($\bigwedge x\ y.$ *tp* $x =$ *tp* $y \Longrightarrow f\ x =$ *None* $\Longrightarrow f\ y =$ *None*) $\Longrightarrow$ *mono* (A::$'a$::*order* $\Rightarrow$ $'b$::*fin-cpo*) $\Longrightarrow$ *mono* B
$\Longrightarrow B$ (*Lfp* ($\lambda a.$ A (*hd* (*the* (f ($B\ a\ \#\ x$)))))) $\le B$ (*Lfp* ($\lambda a.$ A (*hd* (*the* (f ($B\ a\ \#\ y$))))))

thm *mono-fb-t-aux* [*of f x y integer*]

lemma *mono-fb-f*: $f \in \textit{mono-f} \Longrightarrow \textit{le-list}\ x\ y \Longrightarrow (\bigwedge x\ y\ .\ tp\ x\ = tp\ y \Longrightarrow f\ x = \textit{None} \Longrightarrow f\ y = \textit{None})$
$\Longrightarrow\ \textit{fb-t}\ (hd\ (\textit{TI-f}\ f))\ (\lambda a.\ hd\ (\textit{the}\ (f\ (a\ \#\ x)))) \leq \textit{fb-t}\ (hd\ (\textit{TI-f}\ f))\ (\lambda a.\ hd\ (\textit{the}\ (f\ (a\ \#\ y))))$

lemma *fb-mono*: $f \in \textit{mono-f} \Longrightarrow (\bigwedge x\ y\ .\ tp\ x\ = tp\ y \Longrightarrow f\ x = \textit{None} \Longrightarrow f\ y = \textit{None}) \Longrightarrow \textit{fb-f}\ f \in \textit{mono-f}$

lemma *mono-f-mono-a*[*simp*]: $f \in \textit{mono-f} \Longrightarrow f \in\ \textit{has-in-out-type}\ (t\ \#\ t'\ \#\ ts)\ ts' \Longrightarrow tp\ v = ts \Longrightarrow \textit{mono-a}\ (\lambda a\ b.\ hd\ (\textit{the}\ (f\ (b\ \#\ a\ \#\ v))))$

lemma *mono-f-mono-a-b*[*simp*]: $f \in \textit{mono-f} \Longrightarrow f \in\ \textit{has-in-out-type}\ (t\ \#\ t'\ \#\ ts)\ ts' \Longrightarrow tp\ v = ts \Longrightarrow \textit{mono-a}\ (\lambda a\ b.\ hd\ (tl\ (\textit{the}\ (f\ (a\ \#\ b\ \#\ v)))))$

lemma [*simp*]: $\textit{Switch-f}\ x\ y \in \textit{mono-f}$

lemma [*simp*]: $\textit{ID-f}\ x \in \textit{mono-f}$

lemma *has-type-None*: $f \in \textit{has-in-out-type}\ x\ y \Longrightarrow tp\ u = tp\ v \Longrightarrow f\ u = \textit{None} \Longrightarrow f\ v = \textit{None}$

lemma *fb-f-commute*: $f \in \textit{mono-f} \Longrightarrow f \in\ \textit{has-in-out-type}\ (t'\ \#\ t\ \#\ ts)\ (\ t'\ \#\ t\ \#\ ts') \Longrightarrow$
$\textit{fb-f}\ (\textit{fb-f}\ (\textit{par-f}\ (\textit{Switch-f}\ [t']\ [t])\ (\textit{ID-f}\ ts') \circ_m (f \circ_m \textit{par-f}\ (\textit{Switch-f}\ [t]\ [t'])\ (\textit{ID-f}\ ts)))) = (\textit{fb-f}\ (\textit{fb-f}\ f))$

definition $\textit{typed-func} = (\bigcup\ x\ .\ (\bigcup\ y\ .\ \textit{has-in-out-type}\ x\ y)) \cap \textit{mono-f}$

typedef $\textit{func} = \textit{typed-func}$

definition $\textit{fb-func}\ f\ = \textit{Abs-func}\ (\textit{fb-f}\ (\textit{Rep-func}\ f))$

definition $\textit{TI-func}\ f\ = (\textit{TI-f}\ (\textit{Rep-func}\ f))$
definition $\textit{TO-func}\ f\ = (\textit{TO-f}\ (\textit{Rep-func}\ f))$
definition $\textit{ID-func}\ t = \textit{Abs-func}\ (\textit{ID-f}\ t)$

definition $\textit{comp-func}\ f\ g = \textit{Abs-func}\ ((\textit{Rep-func}\ g) \circ_m (\textit{Rep-func}\ f))$

definition $\textit{parallel-func}\ f\ g = \textit{Abs-func}\ (\textit{par-f}\ (\textit{Rep-func}\ f)\ (\textit{Rep-func}\ g))$

definition $\textit{Dup-func}\ x = \textit{Abs-func}\ (\textit{Dup-f}\ x)$

definition $\textit{Sink-func}\ x = \textit{Abs-func}\ (\textit{Sink-f}\ x)$

definition *Switch-func* $x\ y$ = *Abs-func* (*Switch-f* $x\ y$)

lemma [*simp*]: *ID-f* $t \in$ *typed-func*

lemma *map-comp-typed-func*[*simp*]: $f \in$ *typed-func* $\Longrightarrow g \in$ *typed-func* $\Longrightarrow$ *TI-f* g = *TO-f* $f \Longrightarrow (g \circ_m f) \in$ *typed-func*

lemma *par-typed-func*[*simp*]: $f \in$ *typed-func* $\Longrightarrow g \in$ *typed-func* $\Longrightarrow$ *par-f* $f\ g \in$ *typed-func*

lemma *fb-typed-func*[*simp*]: $f \in$ *typed-func* $\Longrightarrow$ *TI-f* $f = t \# x \Longrightarrow$ *TO-f* $f = t \# y \Longrightarrow$ *fb-f* $f \in$ *typed-func*

lemma [*simp*]: *Switch-f* $x\ y \in$ *typed-func*

lemma [*simp*]: *Dup-f* $x \in$ *mono-f*

lemma [*simp*]: *Dup-f* $x \in$ *typed-func*

lemma [*simp*]: *Sink-f* $x \in$ *mono-f*

lemma [*simp*]: *Sink-f* $x \in$ *typed-func*

thm *Rep-func*
thm *Abs-func-inverse*
thm *Rep-func-inverse*

lemma *map-comp-assoc*: $(f \circ_m g) \circ_m h = f \circ_m (g \circ_m h)$

lemma *map-comp-id*: $f \in$ *has-in-out-type* $x\ y \Longrightarrow (f \circ_m$ *ID-f* $x) = f$

lemma *id-map-comp*: $f \in$ *has-in-out-type* $x\ y \Longrightarrow$ (*ID-f* $y \circ_m f) = f$

lemma [*simp*]: $\bigwedge x\ x'$. *tp* $v = x$ @ x' @ $x'' \Longrightarrow$ *pref* (*pref* v (x @ x')) x = *pref* $v\ x$

lemma [*simp*]: $\bigwedge x\ x'$. *tp* $v = x$ @ x' @ $x'' \Longrightarrow$ *suff* (*pref* v (x @ x')) x = *pref* (*suff* $v\ x$) x'

lemma [*simp*]: $\bigwedge x\ x'$. *tp* $v = x$ @ x' @ $x'' \Longrightarrow$*suff* (*suff* $v\ x$) x' = *suff* v (x @ x')

lemma *par-f-assoc*: $f \in$ *has-in-out-type* $x\ y \Longrightarrow g \in$ *has-in-out-type* $x'\ y' \Longrightarrow h \in$ *has-in-out-type* $x''\ y'' \Longrightarrow$
par-f (*par-f* $f\ g$) h = *par-f* f (*par-f* $g\ h$)

lemma $f \in$ *has-in-out-type* $x\ y \Longrightarrow$ *par-f* (*ID-f* []) $f = f$

lemma *id-par-f*: $f \in$ *has-in-out-type* x $y \Longrightarrow$ *par-f* (*ID-f* []) $f = f$

lemma [*simp*]: $\bigwedge x$. *tp* $v = x \Longrightarrow$ *pref* v $x = v$

lemma [*simp*]: $\bigwedge x$. *tp* $v = x \Longrightarrow$ *suff* v $x =$ []

lemma *par-f-id*: $f \in$ *has-in-out-type* x $y \Longrightarrow$ *par-f* f (*ID-f* []) $= f$
lemma [*simp*]: $\bigwedge x$. *tp* $v = x$ @ $y \Longrightarrow$ *pref* v x @ *suff* v $x = v$

lemma [*simp*]: $\bigwedge x$. *tp* $v = x$ @ $x' \Longrightarrow$ *tp* (*pref* v x) $= x$

lemma [*simp*]: $\bigwedge x$. *tp* $v = x$ @ $x' \Longrightarrow$ *tp* (*suff* v x) $= x'$

lemma [*simp*]: $\bigwedge x$. *tp* $u = x \Longrightarrow$ *pref* (u @ v) $x = u$

lemma [*simp*]: $\bigwedge x$. *tp* $u = x \Longrightarrow$ *suff* (u @ v) $x = v$

lemma *par-comp-distrib*: $f \in$ *has-in-out-type* x $y \Longrightarrow g \in$ *has-in-out-type* y $z \Longrightarrow$

$f' \in$ *has-in-out-type* x' $y' \Longrightarrow g' \in$ *has-in-out-type* y' $z' \Longrightarrow$
par-f g $g' \circ_m$ *par-f* f f' = (*par-f* ($g \circ_m f$) ($g' \circ_m f'$))

lemma *TI-f-par*: $f \in$ *typed-func* $\Longrightarrow g \in$ *typed-func* $\Longrightarrow$ *TI-f* (*par-f* f g) $=$ *TI-f* f @ *TI-f* g

lemma *TO-f-par*: $f \in$ *typed-func* $\Longrightarrow g \in$ *typed-func* $\Longrightarrow$ *TO-f* (*par-f* f g) $=$ *TO-f* f @ *TO-f* g

lemma *TI-f-map-comp*[*simp*]: $f \in$ *typed-func* $\Longrightarrow g \in$ *typed-func* $\Longrightarrow$ *TO-f* $g =$ *TI-f* $f \Longrightarrow$ *TI-f* ($f \circ_m g$) $=$ *TI-f* g

lemma *TO-f-map-comp*[*simp*]: $f \in$ *typed-func* $\Longrightarrow g \in$ *typed-func* $\Longrightarrow$ *TO-f* $g =$ *TI-f* $f \Longrightarrow$ *TO-f* ($f \circ_m g$) $=$ *TO-f* f

lemma [*simp*]: *TI-f* (*Sink-f* *ts*) $=$ *ts*

lemma [*simp*]: *TO-f* (*Sink-f* *ts*) $=$ []

lemma *suff-append*: $\bigwedge t$. *tp* $x = t \Longrightarrow$ *suff* (x @ y) $t = y$

lemma [*simp*]: *TI-f* (*Dup-f* x) $= x$

lemma [*simp*]: *TO-f* (*Dup-f* x) $=$ (x @ x)

lemma [*simp*]: *pref* (x @ y) (*tp* x) $= x$

lemma [*simp*]: $\mathit{TO\text{-}f}\ (\mathit{Switch\text{-}f}\ x\ y) = (y\ @\ x)$

lemma [*simp*]: $\mathit{TO\text{-}f}\ (\mathit{ID\text{-}f}\ x) = x$

declare *TO-f-par* [*simp*]

declare *TI-f-par* [*simp*]

lemma [*simp*]: $\bigwedge\ ts\ .\ \ tp\ x = ts\ @\ ts'\ @\ ts'' \Longrightarrow \mathit{pref}\ (\mathit{suff}\ x\ ts)\ ts'\ @\ \mathit{suff}\ x\ (ts\ @\ ts') = \mathit{suff}\ x\ ts$

lemma [*simp*]: $\bigwedge ts\ .\ tp\ x = ts \Longrightarrow\ \ \mathit{suff}\ (x\ @\ y)\ (ts\ @\ ts') = \mathit{suff}\ y\ ts'$

lemma *AAA*: $S\ x \neq \mathit{None} \Longrightarrow tv\ a = t \Longrightarrow tp\ x = \mathit{TI\text{-}f}\ S \Longrightarrow \mathit{the}\ ((\mathit{par\text{-}f}\ (\mathit{ID\text{-}f}\ [t])\ S)\ (a\ \#\ x)) = a\ \#\ \mathit{the}\ (S\ x)$

lemma *AAAb*: $S\ x \neq \mathit{None} \Longrightarrow tv\ a = t \Longrightarrow tp\ x = \mathit{TI\text{-}f}\ S \Longrightarrow ((\mathit{par\text{-}f}\ (\mathit{ID\text{-}f}\ [t])\ S)\ (a\ \#\ x)) = \mathit{Some}\ (a\ \#\ \mathit{the}\ (S\ x))$

lemma *pref-suff-append*: $\bigwedge\ ts\ .\ tp\ x = ts\ @\ ts' \Longrightarrow \mathit{pref}\ x\ ts\ @\ \mathit{suff}\ x\ ts = x$

lemma [*simp*]: $\mathit{Lfp}\ (\lambda\ b.\ a) = a$

lemma [*simp*]: $\mathit{fb\text{-}t}\ (tv\ a)\ (\lambda\ b\ .\ a) = a$

interpretation *func*: *BaseOperation TI-func TO-func ID-func comp-func parallel-func Dup-func Sink-func Switch-func fb-func*
end

7 Diagrams with Named Inputs and Outputs

theory *Diagrams* **imports** *HBDAlgebra*
begin

This file contains the definition and properties for the named input output diagrams

record (*'var*, *'a*) *Dgr* =
  In:: *'var list*
  Out:: *'var list*
  Trs:: *'a*

context *BaseOperationFeedbacklessVars*
begin
definition $\mathit{Var}\ A\ B = (\mathit{Out}\ A) \otimes (\mathit{In}\ B)$

definition *io-diagram* $A = (\mathit{TVs}\ (\mathit{In}\ A) = \mathit{TI}\ (\mathit{Trs}\ A) \wedge \mathit{TVs}\ (\mathit{Out}\ A) = \mathit{TO}\ (\mathit{Trs}\ A) \wedge \mathit{distinct}\ (\mathit{In}\ A) \wedge \mathit{distinct}\ (\mathit{Out}\ A))$

definition *Comp* :: $('var, 'a)$ *Dgr* $\Rightarrow$ $('var, 'a)$ *Dgr* $\Rightarrow$ $('var, 'a)$ *Dgr* (**infixl** ;; *70*) **where**
A ;; $B = (let\ I = In\ B \ominus Var\ A\ B\ in\ let\ O' = Out\ A \ominus Var\ A\ B\ in$
$(\!|In = (In\ A) \oplus I,\ Out = O'\ @\ Out\ B,$
$Trs = [(In\ A) \oplus I \rightsquigarrow In\ A\ @\ I\ ]\ oo\ Trs\ A \parallel [I \rightsquigarrow I]\ oo\ [Out\ A\ @\ I \rightsquigarrow O'\ @\ In\ B]\ \ oo\ ([O' \rightsquigarrow O'] \parallel Trs\ B)\ |\!))$

lemma *io-diagram-Comp*: $io\text{-}diagram\ A \Longrightarrow io\text{-}diagram\ B$
$\Longrightarrow set\ (Out\ A \ominus In\ B) \cap set\ (Out\ B) = \{\} \Longrightarrow io\text{-}diagram\ (A\ ;;\ B)$

lemma *Comp-in-disjoint*:
assumes $io\text{-}diagram\ A$
and $io\text{-}diagram\ B$
and $set\ (In\ A) \cap set\ (In\ B) = \{\}$
shows $A\ ;;\ B = (let\ I = In\ B \ominus Var\ A\ B\ in\ let\ O' = Out\ A \ominus Var\ A\ B\ in$
$(\!|In = (In\ A)\ @\ I,\ Out = O'\ @\ Out\ B,\ Trs = Trs\ A \parallel [I \rightsquigarrow I]\ oo\ [Out\ A\ @\ I \rightsquigarrow O'\ @\ In\ B]\ \ oo\ ([O' \rightsquigarrow O'] \parallel Trs\ B)\ |\!))$

lemma *Comp-full*: $io\text{-}diagram\ A \Longrightarrow io\text{-}diagram\ B \Longrightarrow Out\ A = In\ B \Longrightarrow$
$A\ ;;\ B = (\!|In = In\ A,\ Out = Out\ B,\ Trs = Trs\ A\ oo\ Trs\ B\ |\!)$

lemma *Comp-in-out*: $io\text{-}diagram\ A \Longrightarrow io\text{-}diagram\ B \Longrightarrow set\ (Out\ A) \subseteq set\ (In\ B) \Longrightarrow$
$A\ ;;\ B = (let\ I = diff\ (In\ B)\ (Var\ A\ B)\ in\ let\ O' = diff\ (Out\ A)\ (Var\ A\ B)\ in$
$(\!|In = In\ A \oplus I,\ Out = Out\ B,\ Trs = [In\ A \oplus I \rightsquigarrow In\ A\ @\ I\ ]\ oo\ Trs\ A \parallel [I \rightsquigarrow I]\ oo\ [Out\ A\ @\ I \rightsquigarrow In\ B]\ oo\ Trs\ B\ |\!))$

lemma *Comp-assoc-new*: $io\text{-}diagram\ A \Longrightarrow io\text{-}diagram\ B \Longrightarrow io\text{-}diagram\ C \Longrightarrow$
$set\ (Out\ A \ominus In\ B) \cap set\ (Out\ B) = \{\} \Longrightarrow\ set\ (Out\ A \otimes In\ B) \cap set\ (In\ C) = \{\}$
$\Longrightarrow A\ ;;\ B\ ;;\ C = A\ ;;\ (B\ ;;\ C)$

lemma *Comp-assoc-a*: $io\text{-}diagram\ A \Longrightarrow io\text{-}diagram\ B \Longrightarrow io\text{-}diagram\ C \Longrightarrow$
$set\ (In\ B) \cap set\ (In\ C) = \{\} \Longrightarrow$
$set\ (Out\ A) \cap set\ (Out\ B) = \{\} \Longrightarrow$
$A\ ;;\ B\ ;;\ C = A\ ;;\ (B\ ;;\ C)$

definition *Parallel* :: $('var, 'a)$ *Dgr* $\Rightarrow$ $('var, 'a)$ *Dgr* $\Rightarrow$ $('var, 'a)$ *Dgr* (**infixl** ||| *80*) **where**
$A\ |||\ B = (\!|In = In\ A \oplus In\ B,\ Out = Out\ A\ @\ Out\ B,\ Trs = [In\ A \oplus In\ B \rightsquigarrow In\ A\ @\ In\ B]\ oo\ (Trs\ A \parallel Trs\ B)\ |\!)$

lemma *io-diagram-Parallel*: $io\text{-}diagram\ A \Longrightarrow io\text{-}diagram\ B\ \Longrightarrow set\ (Out\ A)$

$\cap$ *set* (*Out B*) = {} $\Longrightarrow$ *io-diagram* (*A* ||| *B*)

lemma *Parallel-indep*: *io-diagram A* $\Longrightarrow$ *io-diagram B* $\Longrightarrow$ *set* (*In A*) $\cap$ *set* (*In B*) = {} $\Longrightarrow$
A ||| *B* = (|*In* = *In A* @ *In B*, *Out* = *Out A* @ *Out B*, *Trs* = (*Trs A* ‖ *Trs B*) |)

lemma *Parallel-assoc-gen*: *io-diagram A* $\Longrightarrow$ *io-diagram B* $\Longrightarrow$ *io-diagram C* $\Longrightarrow$
A ||| *B* ||| *C* = *A* ||| (*B* ||| *C*)

definition *VarFB A* = *Var A A*
definition *InFB A*= *In A* $\ominus$ *VarFB A*
definition *OutFB A* = *Out A* $\ominus$ *VarFB A*

definition *FB* :: ('*var*, '*a*) *Dgr* $\Rightarrow$ ('*var*, '*a*) *Dgr* **where**
FB A = (*let I* = *In A* $\ominus$ *Var A A in let O'* = *Out A* $\ominus$ *Var A A in*
(|*In* = *I*, *Out* = *O'*, *Trs* = (*fb* ^^ (*length* (*Var A A*))) ([*Var A A* @ *I* $\rightsquigarrow$ *In A*] *oo Trs A oo* [*Out A* $\rightsquigarrow$ *Var A A* @ *O'*]) |))

lemma *Type-ok-FB*: *io-diagram A* $\Longrightarrow$ *io-diagram* (*FB A*)

lemma *perm-var-Par*: *io-diagram A* $\Longrightarrow$ *io-diagram B* $\Longrightarrow$ *set* (*In A*) $\cap$ *set* (*In B*) = {}
$\Longrightarrow$ *perm* (*Var* (*A* ||| *B*) (*A* ||| *B*)) (*Var A A* @ *Var B B* @ *Var A B* @ *Var B A*)

lemma *distinct-Parallel-Var*[*simp*]: *io-diagram A* $\Longrightarrow$ *io-diagram B*
$\Longrightarrow$ *set* (*Out A*) $\cap$ *set* (*Out B*) = {} $\Longrightarrow$ *distinct* (*Var* (*A* ||| *B*) (*A* ||| *B*))

lemma *distinct-Parallel-In*[*simp*]: *io-diagram A* $\Longrightarrow$ *io-diagram B* $\Longrightarrow$ *distinct* (*In* (*A* ||| *B*))

lemma *drop-assumption*: *p* $\Longrightarrow$ *True*

lemma *Dgr-eq*: *In A* = *x* $\Longrightarrow$ *Out A* = *y* $\Longrightarrow$ *Trs A* = *S* $\Longrightarrow$ (|*In* = *x*, *Out* = *y*, *Trs* = *S*|) = *A*

lemma *Var-FB*[*simp*]: *Var* (*FB A*) (*FB A*) = []

theorem *FB-idemp*: *io-diagram A* $\Longrightarrow$ *FB* (*FB A*) = *FB A*

definition *VarSwitch* :: $'var\ list \Rightarrow 'var\ list \Rightarrow ('var, 'a)\ Dgr$ ([[- $\rightsquigarrow$ -]]) **where**
VarSwitch $x\ y = (\!| In = x,\ Out = y,\ Trs = [x \rightsquigarrow y] |\!)$

definition *in-equiv* $A\ B = (perm\ (In\ A)\ (In\ B) \wedge Trs\ A = [In\ A \rightsquigarrow In\ B]\ oo\ Trs\ B \wedge Out\ A = Out\ B)$

definition *out-equiv* $A\ B = (perm\ (Out\ A)\ (Out\ B) \wedge Trs\ A = Trs\ B\ oo\ [Out\ B \rightsquigarrow Out\ A] \wedge In\ A = In\ B)$

definition *in-out-equiv* $A\ B = (perm\ (In\ A)\ (In\ B) \wedge perm\ (Out\ A)\ (Out\ B) \wedge Trs\ A = [In\ A \rightsquigarrow In\ B]\ oo\ Trs\ B\ oo\ [Out\ B \rightsquigarrow Out\ A])$

lemma *in-equiv-io-diagram*: *in-equiv* $A\ B \Longrightarrow$ *io-diagram* $B \Longrightarrow$ *io-diagram* A

lemma *in-out-equiv-io-diagram*: *in-out-equiv* $A\ B \Longrightarrow$ *io-diagram* $B \Longrightarrow$ *io-diagram* A

lemma *in-equiv-sym*: *io-diagram* $B \Longrightarrow$ *in-equiv* $A\ B \Longrightarrow$ *in-equiv* $B\ A$

lemma *in-equiv-eq*: *io-diagram* $A \Longrightarrow A = B \Longrightarrow$ *in-equiv* $A\ B$

lemma [*simp*]: *io-diagram* $A \Longrightarrow [In\ A \rightsquigarrow In\ A]\ oo\ Trs\ A\ oo\ [Out\ A \rightsquigarrow Out\ A] = Trs\ A$

lemma *in-equiv-tran*: *io-diagram* $C \Longrightarrow$ *in-equiv* $A\ B \Longrightarrow$ *in-equiv* $B\ C \Longrightarrow$ *in-equiv* $A\ C$

lemma *in-out-equiv-refl*: *io-diagram* $A \Longrightarrow$ *in-out-equiv* $A\ A$

lemma *in-out-equiv-sym*: *io-diagram* $A \Longrightarrow$ *io-diagram* $B \Longrightarrow$ *in-out-equiv* $A\ B \Longrightarrow$ *in-out-equiv* $B\ A$

lemma *in-out-equiv-tran*: *io-diagram* $A \Longrightarrow$ *io-diagram* $B \Longrightarrow$ *io-diagram* $C \Longrightarrow$ *in-out-equiv* $A\ B \Longrightarrow$ *in-out-equiv* $B\ C \Longrightarrow$ *in-out-equiv* $A\ C$

lemma [*simp*]: *distinct* $(Out\ A) \Longrightarrow$ *distinct* $(Var\ A\ B)$

lemma [*simp*]: *set* $(Var\ A\ B) \subseteq$ *set* $(Out\ A)$
lemma [*simp*]: *set* $(Var\ A\ B) \subseteq$ *set* $(In\ B)$

lemmas *fb-indep-sym* = *fb-indep* [*THEN sym*]

declare *length-TVs* [*simp*]

end

primrec *op-list* :: $'a \Rightarrow ('a \Rightarrow 'a \Rightarrow 'a) \Rightarrow 'a\ list \Rightarrow 'a$ **where**
op-list e opr $[] = e$ |
op-list e opr $(a \# x) =$ *opr a* $($*op-list e opr x*$)$

primrec *inter-set* :: $'a\ list \Rightarrow 'a\ set \Rightarrow 'a\ list$ **where**
inter-set $[]\ X = []$ |
inter-set $(x \# xs)\ X = ($*if* $x \in X$ *then* $x \#$ *inter-set xs X else inter-set xs X*$)$

lemma *list-inter-set*: $x \otimes y =$ *inter-set x* $($*set y*$)$

fun *map2* :: $('a \Rightarrow 'b \Rightarrow bool) \Rightarrow 'a\ list \Rightarrow 'b\ list \Rightarrow bool$ **where**
map2 f $[]\ [] =$ *True* |
map2 f $(a \# x)\ (b \# y) = (f\ a\ b \wedge$ *map2 f x y*$)$ |
map2 - - - = *False*

thm *map-def*

context *BaseOperationFeedbacklessVars*
begin
definition *ParallelId* :: $('var, 'a)$ *Dgr* $(\square)$
where $\square = (\lvert In = [],\ Out = [],\ Trs = ID\ [] \rvert)$

lemma [*simp*]: *Out* $\square = []$

lemma [*simp*]: *In* $\square = []$

lemma [*simp*]: *Trs* $\square = ID\ []$

lemma *ParallelId-right*[*simp*]: *io-diagram A* $\Longrightarrow A\ |||\ \square = A$

lemma *ParallelId-left*: *io-diagram A* $\Longrightarrow \square\ |||\ A = A$

definition *parallel-list* = *op-list* $(ID\ [])\ (\|)$

definition *Parallel-list* = *op-list* $\square\ (|||)$

lemma [*simp*]: *Parallel-list* $[] = \square$

definition *io-distinct As* = $($*distinct* $($*concat* $($*map In As*$)) \wedge$ *distinct* $($*concat* $($*map Out As*$)) \wedge (\forall\ A \in$ *set As* . *io-diagram A*$))$

definition *io-rel A* = *set* $($*Out A*$) \times$ *set* $($*In A*$)$

definition *IO-Rel As* = $\bigcup$ $($*set* $($*map io-rel As*$))$

definition *out A* = *hd* $($*Out A*$)$

definition $Type\text{-}OK\ As = ((\forall\ B \in set\ As\ .\ io\text{-}diagram\ B \wedge length\ (Out\ B) = 1)$
$\wedge\ distinct\ (concat\ (map\ Out\ As)))$

lemma *concat-map-out*: $(\forall\ A \in set\ As\ .\ length\ (Out\ A) = 1) \Longrightarrow concat\ (map\ Out\ As) = map\ out\ As$

lemma *Type-OK-simp*: $Type\text{-}OK\ As = ((\forall\ B \in set\ As\ .\ io\text{-}diagram\ B \wedge length\ (Out\ B) = 1) \wedge distinct\ (map\ out\ As))$

definition $single\text{-}out\ A = (io\text{-}diagram\ A \wedge length\ (Out\ A) = 1)$

definition $CompA :: ('var, 'a)\ Dgr \Rightarrow ('var, 'a)\ Dgr \Rightarrow ('var, 'a)\ Dgr$ (**infixl** $\rhd$ 75) **where**

$A \rhd B = (if\ out\ A \in set\ (In\ B)\ then\ A\ ;;\ B\ else\ B)$

definition $internal\ As = \{x\ .\ (\exists\ A \in set\ As\ .\ \exists\ B \in set\ As\ .\ x \in set\ (Out\ A) \wedge x \in set\ (In\ B))\}$

primrec $get\text{-}comp\text{-}out :: 'var \Rightarrow ('var, 'a)\ Dgr\ list \Rightarrow ('var, 'a)\ Dgr$ **where**
$get\text{-}comp\text{-}out\ x\ [] = (\!|In = [x],\ Out = [x],\ Trs = [\ [x] \leadsto [x]\ ]|\!)\ |$
$get\text{-}comp\text{-}out\ x\ (A\ \#\ As) = (if\ x \in set\ (Out\ A)\ then\ A\ else\ get\text{-}comp\text{-}out\ x\ As)$

primrec $get\text{-}other\text{-}out :: 'c \Rightarrow ('c, 'd)\ Dgr\ list \Rightarrow ('c, 'd)\ Dgr\ list$ **where**
$get\text{-}other\text{-}out\ x\ [] = []\ |$
$get\text{-}other\text{-}out\ x\ (A\ \#\ As) = (if\ x \in set\ (Out\ A)\ then\ get\text{-}other\text{-}out\ x\ As\ else\ A\ \#\ get\text{-}other\text{-}out\ x\ As)$

definition $fb\text{-}less\text{-}step\ A\ As = map\ (CompA\ A)\ As$

definition $fb\text{-}out\text{-}less\text{-}step\ x\ As = fb\text{-}less\text{-}step\ (get\text{-}comp\text{-}out\ x\ As)\ (get\text{-}other\text{-}out\ x\ As)$

primrec $fb\text{-}less :: 'var\ list \Rightarrow ('var, 'a)\ Dgr\ list \Rightarrow ('var, 'a)\ Dgr\ list$ **where**
$fb\text{-}less\ []\ As = As\ |$
$fb\text{-}less\ (x\ \#\ xs)\ As = fb\text{-}less\ xs\ (fb\text{-}out\text{-}less\text{-}step\ x\ As)$

lemma [*simp*]: $VarFB\ \square = []$
lemma [*simp*]: $InFB\ \square = []$
lemma [*simp*]: $OutFB\ \square = []$

definition $loop\text{-}free\ As = (\forall\ x\ .\ (x,x) \notin (IO\text{-}Rel\ As)^{+})$

lemma [*simp*]: $Parallel\text{-}list\ (A\ \#\ As) = (A\ |||\ Parallel\text{-}list\ As)$

lemma [*simp*]: $Out\ (A\ |||\ B) = Out\ A\ @\ Out\ B$

lemma [*simp*]: $In\ (A\ |||\ B) = In\ A \oplus In\ B$

lemma *Type-OK-cons*: $Type\text{-}OK\ (A\ \#\ As) = (io\text{-}diagram\ A \wedge length\ (Out\ A) = 1 \wedge set\ (Out\ A) \cap (\bigcup a{\in}set\ As.\ set\ (Out\ a)) = \{\} \wedge Type\text{-}OK\ As)$

lemma *Out-Parallel*: $Out\ (Parallel\text{-}list\ As) = concat\ (map\ Out\ As)$

lemma *internal-cons*: $internal\ (A\ \#\ As) = \{x.\ x \in set\ (Out\ A) \wedge (x \in set\ (In\ A) \vee (\exists B{\in}set\ As.\ x \in set\ (In\ B)))\} \cup \{x\ .\ (\exists Aa{\in}set\ As.\ x \in set\ (Out\ Aa) \wedge (x \in set\ (In\ A)))\}$
$\cup\ internal\ As$

lemma *Out-out*: $length\ (Out\ A) = Suc\ 0 \Longrightarrow Out\ A = [out\ A]$

lemma *Type-OK-out*: $Type\text{-}OK\ As \Longrightarrow A \in set\ As \Longrightarrow Out\ A = [out\ A]$

lemma *In-Parallel*: $In\ (Parallel\text{-}list\ As) = op\text{-}list\ []\ (\oplus)\ (map\ In\ As)$

lemma [*simp*]: $set\ (op\text{-}list\ []\ (\oplus)\ xs) = \bigcup\ set\ (map\ set\ xs)$

lemma *internal-VarFB*: $Type\text{-}OK\ As \Longrightarrow internal\ As = set\ (VarFB\ (Parallel\text{-}list\ As))$

lemma *map-Out-fb-less-step*: $length\ (Out\ A) = 1 \Longrightarrow map\ Out\ (fb\text{-}less\text{-}step\ A\ As) = map\ Out\ As$

lemma *mem-get-comp-out*: $Type\text{-}OK\ As \Longrightarrow A \in set\ As \Longrightarrow get\text{-}comp\text{-}out\ (out\ A)\ As = A$

lemma *map-Out-fb-out-less-step*: $A \in set\ As \Longrightarrow Type\text{-}OK\ As \Longrightarrow a = out\ A \Longrightarrow map\ Out\ (fb\text{-}out\text{-}less\text{-}step\ a\ As) = map\ Out\ (get\text{-}other\text{-}out\ a\ As)$

lemma [*simp*]: $Type\text{-}OK\ (A\ \#\ As) \Longrightarrow Type\text{-}OK\ As$

lemma *Type-OK-Out*: $Type\text{-}OK\ (A\ \#\ As) \Longrightarrow Out\ A = [out\ A]$

lemma *concat-map-Out-get-other-out*: $Type\text{-}OK\ As \Longrightarrow concat\ (map\ Out\ (get\text{-}other\text{-}out\ a\ As)) = (concat\ (map\ Out\ As) \ominus [a])$

thm *Out-out*

lemma *VarFB-cons-out*: *Type-OK As* $\Longrightarrow$ *VarFB* (*Parallel-list As*) = *a* # *L* $\Longrightarrow$ $\exists$ *A* $\in$ *set As* . *out A* = *a*

lemma *VarFB-cons-out-In*: *Type-OK As* $\Longrightarrow$ *VarFB* (*Parallel-list As*) = *a* # *L* $\Longrightarrow$ $\exists$ *B* $\in$ *set As* . *a* $\in$ *set* (*In B*)

lemma *AAA-a*: *Type-OK* (*A* # *As*) $\Longrightarrow$ *A* $\notin$ *set As*

lemma *AAA-b*: ($\forall$ *A* $\in$ *set As*. *a* $\notin$ *set* (*Out A*)) $\Longrightarrow$ *get-other-out a As* = *As*

lemma *AAA-d*: *Type-OK* (*A* # *As*) $\Longrightarrow$ $\forall$ *Aa*$\in$*set As*. *out A* $\neq$ *out Aa*

lemma *mem-get-other-out*: *Type-OK As* $\Longrightarrow$ *A* $\in$ *set As* $\Longrightarrow$ *get-other-out* (*out A*) *As* = (*As* $\ominus$ [*A*])

lemma *In-CompA*: *In* (*A* $\rhd$ *B*) = (*if out A* $\in$ *set* (*In B*) *then In A* $\oplus$ (*In B* $\ominus$ *Out A*) *else In B*)

lemma *union-set-In-CompA*: $\bigwedge$ *B* . *length* (*Out A*) = *1* $\Longrightarrow$ *B* $\in$ *set As* $\Longrightarrow$ *out A* $\in$ *set* (*In B*)
$\Longrightarrow$ ($\bigcup$*x*$\in$*set As*. *set* (*In* (*CompA A x*))) = *set* (*In A*) $\cup$ (($\bigcup$ *B* $\in$ *set As* . *set* (*In B*)) $-$ {*out A*})

lemma *BBBB-e*: *Type-OK As* $\Longrightarrow$ *VarFB* (*Parallel-list As*) = *out A* # *L* $\Longrightarrow$ *A* $\in$ *set As* $\Longrightarrow$ *out A* $\notin$ *set L*

lemma *BBBB-f*: *loop-free As* $\Longrightarrow$
Type-OK As $\Longrightarrow$ *A* $\in$ *set As* $\Longrightarrow$ *B* $\in$ *set As* $\Longrightarrow$ *out A* $\in$ *set* (*In B*) $\Longrightarrow$ *B* $\neq$ *A*

thm *union-set-In-CompA*

lemma [*simp*]: *x* $\in$ *set* (*Out* (*get-comp-out x As*))

lemma *comp-out-in*: *A* $\in$ *set As* $\Longrightarrow$ *a* $\in$ *set* (*Out A*) $\Longrightarrow$ (*get-comp-out a As*) $\in$ *set As*

lemma [*simp*]: *a* $\in$ *internal As* $\Longrightarrow$ *get-comp-out a As* $\in$ *set As*

lemma *out-CompA*: *length* (*Out A*) = *1* $\Longrightarrow$ *out* (*CompA A B*) = *out B*

lemma *Type-OK-loop-free-elem*: *Type-OK As* $\Longrightarrow$ *loop-free As* $\Longrightarrow$ *A* $\in$ *set As* $\Longrightarrow$ *out A* $\notin$ *set* (*In A*)

lemma *BBB-a*: *length* (*Out A*) = *1* $\Longrightarrow$ *Out* (*CompA A B*) = *Out B*

lemma *BBB-b*: *length* (*Out A*) = *1* $\Longrightarrow$ *map* (*Out* $\circ$ *CompA A*) *As* = *map Out As*

lemma *VarFB-fb-out-less-step-gen*:
assumes *loop-free As*
assumes *Type-OK As*
and *internal-a*: *a* $\in$ *internal As*
shows *VarFB* (*Parallel-list* (*fb-out-less-step a As*)) = (*VarFB* (*Parallel-list As*)) $\ominus$ [*a*]

thm *internal-VarFB*
thm *VarFB-fb-out-less-step-gen*

lemma *VarFB-fb-out-less-step*: *loop-free As* $\Longrightarrow$ *Type-OK As* $\Longrightarrow$ *VarFB* (*Parallel-list As*) = *a* # *L* $\Longrightarrow$ *VarFB* (*Parallel-list* (*fb-out-less-step a As*)) = *L*

lemma *Parallel-list-cons*:*Parallel-list* (*a* # *As*) = *a* ||| *Parallel-list As*

lemma *io-diagram-parallel-list*: *Type-OK As* $\Longrightarrow$ *io-diagram* (*Parallel-list As*)

lemma *BBB-c*: *distinct* (*map f As*) $\Longrightarrow$ *distinct* (*map f* (*As* $\ominus$ *Bs*))

lemma *io-diagram-CompA*: *io-diagram A* $\Longrightarrow$ *length* (*Out A*) = *1* $\Longrightarrow$ *io-diagram B* $\Longrightarrow$ *io-diagram* (*CompA A B*)

lemma *Type-OK-fb-out-less-step-aux*: *Type-OK As* $\Longrightarrow$ *A* $\in$ *set As* $\Longrightarrow$ *Type-OK* (*fb-less-step A* (*As* $\ominus$ [*A*]))

thm *VarFB-cons-out*

theorem *Type-OK-fb-out-less-step-new*: *Type-OK As* $\Longrightarrow$
a $\in$ *internal As* $\Longrightarrow$
Bs = *fb-out-less-step a As* $\Longrightarrow$ *Type-OK Bs*

theorem *Type-OK-fb-out-less-step*: *loop-free As* $\Longrightarrow$ *Type-OK As* $\Longrightarrow$
VarFB (*Parallel-list As*) = *a* # *L* $\Longrightarrow$ *Bs* = *fb-out-less-step a As* $\Longrightarrow$

Type-OK Bs

lemma *perm-FB-Parallel*[*simp*]: *loop-free As* $\Longrightarrow$ *Type-OK As*
$\Longrightarrow$ *VarFB* (*Parallel-list As*) = *a* # *L* $\Longrightarrow$ *Bs* = *fb-out-less-step a As*
$\Longrightarrow$ *perm* (*In* (*FB* (*Parallel-list As*))) (*In* (*FB* (*Parallel-list Bs*)))

lemma [*simp*]: *loop-free As* $\Longrightarrow$ *Type-OK As* $\Longrightarrow$
VarFB (*Parallel-list As*) = *a* # *L* $\Longrightarrow$
Out (*FB* (*Parallel-list* (*fb-out-less-step a As*))) = *Out* (*FB* (*Parallel-list As*))

lemma *TI-Parallel-list*: ($\forall$ *A* $\in$ *set As* . *io-diagram A*) $\Longrightarrow$ *TI* (*Trs* (*Parallel-list As*)) = *TVs* (*op-list* [] ($\oplus$) (*map In As*))

lemma *TO-Parallel-list*: ($\forall$ *A* $\in$ *set As* . *io-diagram A*) $\Longrightarrow$ *TO* (*Trs* (*Parallel-list As*)) = *TVs* (*concat* (*map Out As*))

lemma *fbtype-aux*: (*Type-OK As*) $\Longrightarrow$ *loop-free As* $\Longrightarrow$ *VarFB* (*Parallel-list As*) = *a* # *L* $\Longrightarrow$
fbtype ([*L* @ (*In* (*Parallel-list* (*fb-out-less-step a As*)) $\ominus$ *L*) $\rightsquigarrow$ *In* (*Parallel-list* (*fb-out-less-step a As*))] *oo Trs* (*Parallel-list* (*fb-out-less-step a As*)) *oo*
[*Out* (*Parallel-list* (*fb-out-less-step a As*)) $\rightsquigarrow$ *L* @ (*Out* (*Parallel-list* (*fb-out-less-step a As*)) $\ominus$ *L*)])
(*TVs L*) (*TO* [*In* (*Parallel-list As*) $\ominus$ *a* # *L* $\rightsquigarrow$ *In* (*Parallel-list* (*fb-out-less-step a As*)) $\ominus$ *L*]) (*TVs* (*Out* (*Parallel-list* (*fb-out-less-step a As*)) $\ominus$ *L*))

lemma *fb-indep-left-a*: *fbtype S tsa* (*TO A*) *ts* $\Longrightarrow$ *A oo* (*fb*^^(*length tsa*)) *S* = (*fb*^^(*length tsa*)) ((*ID tsa* $\|$ *A*) *oo S*)

lemma *parallel-list-cons*: *parallel-list* (*A* # *As*) = *A* $\|$ *parallel-list As*

lemma *TI-parallel-list*: ($\forall$ *A* $\in$ *set As* . *io-diagram A*) $\Longrightarrow$ *TI* (*parallel-list* (*map Trs As*)) = *TVs* (*concat* (*map In As*))

lemma *TO-parallel-list*: ($\forall$ *A* $\in$ *set As* . *io-diagram A*) $\Longrightarrow$ *TO* (*parallel-list* (*map Trs As*)) = *TVs* (*concat* (*map Out As*))

lemma *Trs-Parallel-list-aux-a*: *Type-OK As* $\Longrightarrow$ *io-diagram a* $\Longrightarrow$
[*In a* $\oplus$ *In* (*Parallel-list As*) $\rightsquigarrow$ *In a* @ *In* (*Parallel-list As*)] *oo Trs a* $\|$ ([*In* (*Parallel-list As*) $\rightsquigarrow$ *concat* (*map In As*)] *oo parallel-list* (*map Trs As*)) =
[*In a* $\oplus$ *In* (*Parallel-list As*) $\rightsquigarrow$ *In a* @ *In* (*Parallel-list As*)] *oo* ([*In a* $\rightsquigarrow$ *In a*] $\|$ [*In* (*Parallel-list As*) $\rightsquigarrow$ *concat* (*map In As*)] *oo Trs a* $\|$ *parallel-list* (*map*

Trs As))

lemma *Trs-Parallel-list-aux-b* :*distinct* $x \Longrightarrow$ *distinct* $y \Longrightarrow$ *set* $z \subseteq$ *set* $y \Longrightarrow [x \oplus y \rightsquigarrow x @ y]$ *oo* $[x \rightsquigarrow x] \parallel [y \rightsquigarrow z] = [x \oplus y \rightsquigarrow x @ z]$

lemma *Trs-Parallel-list*: *Type-OK As* $\Longrightarrow$ *Trs* (*Parallel-list As*) = [*In* (*Parallel-list As*) $\rightsquigarrow$ *concat* (*map In As*)] *oo parallel-list* (*map Trs As*)

lemma *CompA-Id*[*simp*]: $A \rhd \square = \square$

lemma *io-diagram-ParallelId*[*simp*]: *io-diagram* $\square$

lemma *in-equiv-aux-a* :*distinct* $x \Longrightarrow$ *distinct* $y \Longrightarrow$ *set* $z \subseteq$ *set* $x \Longrightarrow [x \oplus y \rightsquigarrow x @ y]$ *oo* $[x \rightsquigarrow z] \parallel [y \rightsquigarrow y] = [x \oplus y \rightsquigarrow z @ y]$

lemma *in-equiv-Parallel-aux-d*: *distinct* $x \Longrightarrow$ *distinct* $y \Longrightarrow$ *set* $u \subseteq$ *set* $x \Longrightarrow$ *perm* y v
$\Longrightarrow [x \oplus y \rightsquigarrow x @ v]$ *oo* $[x \rightsquigarrow u] \parallel [v \rightsquigarrow v] = [x \oplus y \rightsquigarrow u @ v]$

lemma *comp-par-switch-subst*: *distinct* $x \Longrightarrow$ *distinct* $y \Longrightarrow$ *set* $u \subseteq$ *set* $x \Longrightarrow$ *set* $v \subseteq$ *set* y
$\Longrightarrow [x \oplus y \rightsquigarrow x @ y]$ *oo* $[x \rightsquigarrow u] \parallel [y \rightsquigarrow v] = [x \oplus y \rightsquigarrow u @ v]$

lemma *in-equiv-Parallel-aux-b* :*distinct* $x \Longrightarrow$ *distinct* $y \Longrightarrow$ *perm* u $x \Longrightarrow$ *perm* y $v \Longrightarrow [x \oplus y \rightsquigarrow x @ y]$ *oo* $[x \rightsquigarrow u] \parallel [y \rightsquigarrow v] = [x \oplus y \rightsquigarrow u @ v]$

lemma [*simp*]: *set* $x \subseteq$ *set* $(x \oplus y)$

lemma [*simp*]: *set* $y \subseteq$ *set* $(x \oplus y)$

declare *distinct-addvars* [*simp*]

lemma *in-equiv-Parallel*: *io-diagram* $B \Longrightarrow$ *io-diagram* $B' \Longrightarrow$ *in-equiv* A $B \Longrightarrow$ *in-equiv* A' $B' \Longrightarrow$ *in-equiv* $(A ||| A')$ $(B ||| B')$

thm *local.BBB-a*

lemma *map-Out-CompA*: *length* (*Out* A) = $1 \Longrightarrow$ *map* (*out* $\circ$ *CompA* A) *As*

$= map\ out\ As$

lemma *CompA-in*[*simp*]: $out\ A \in set\ (In\ B) \Longrightarrow A \rhd B = A\ ;;\ B$

lemma *CompA-not-in*[*simp*]: $out\ A \notin set\ (In\ B) \Longrightarrow A \rhd B = B$

lemma *in-equiv-CompA-Parallel-a*: $deterministic\ (Trs\ A) \Longrightarrow length\ (Out\ A) = 1 \Longrightarrow io\text{-}diagram\ A \Longrightarrow io\text{-}diagram\ B \Longrightarrow io\text{-}diagram\ C$
$\Longrightarrow out\ A \in set\ (In\ B) \Longrightarrow out\ A \in set\ (In\ C)$
$\Longrightarrow in\text{-}equiv\ ((A \rhd B)\ |||\ (A \rhd C))\ (A \rhd (B\ |||\ C))$

lemma *in-equiv-CompA-Parallel-c*: $length\ (Out\ A) = 1 \Longrightarrow io\text{-}diagram\ A \Longrightarrow io\text{-}diagram\ B \Longrightarrow io\text{-}diagram\ C \Longrightarrow out\ A \notin set\ (In\ B) \Longrightarrow out\ A \in set\ (In\ C) \Longrightarrow$
$in\text{-}equiv\ (CompA\ A\ B\ |||\ CompA\ A\ C)\ (CompA\ A\ (B\ |||\ C))$

lemmas *distinct-addvars distinct-diff*

lemma *io-diagram-distinct*: **assumes** *A*: *io-diagram A* **shows** [*simp*]: $distinct\ (In\ A)$
and [*simp*]: $distinct\ (Out\ A)$ **and** [*simp*]: $TI\ (Trs\ A) = TVs\ (In\ A)$
and [*simp*]: $TO\ (Trs\ A) = TVs\ (Out\ A)$

declare *Subst-not-in-a* [*simp*]
declare *Subst-not-in* [*simp*]

lemma [*simp*]: $set\ x' \cap set\ z = \{\} \Longrightarrow TVs\ x = TVs\ y \Longrightarrow TVs\ x' = TVs\ y' \Longrightarrow Subst\ (x\ @\ x')\ (y\ @\ y')\ z = Subst\ x\ y\ z$

lemma [*simp*]: $set\ x \cap set\ z = \{\} \Longrightarrow TVs\ x = TVs\ y \Longrightarrow TVs\ x' = TVs\ y' \Longrightarrow Subst\ (x\ @\ x')\ (y\ @\ y')\ z = Subst\ x'\ y'\ z$

lemma [*simp*]: $set\ x \cap set\ z = \{\} \Longrightarrow TVs\ x = TVs\ y \Longrightarrow Subst\ x\ y\ z = z$

lemma [*simp*]: $distinct\ x \Longrightarrow TVs\ x = TVs\ y \Longrightarrow Subst\ x\ y\ x = y$

lemma $TVs\ x = TVs\ y \Longrightarrow length\ x = length\ y$

thm *length-TVs*

lemma *in-equiv-switch-Parallel*: $io\text{-}diagram\ A \Longrightarrow io\text{-}diagram\ B \Longrightarrow set\ (Out\ A) \cap set\ (Out\ B) = \{\} \Longrightarrow$
$in\text{-}equiv\ (A\ |||\ B)\ ((B\ |||\ A)\ ;;\ [[\ Out\ B\ @\ Out\ A \rightsquigarrow Out\ A\ @\ Out\ B]])$

lemma *in-out-equiv-Parallel*: $io\text{-}diagram\ A \Longrightarrow io\text{-}diagram\ B \Longrightarrow set\ (Out\ A) \cap set\ (Out\ B) = \{\} \Longrightarrow in\text{-}out\text{-}equiv\ (A\ |||\ B)\ (B\ |||\ A)$

declare *Subst-eq* [*simp*]

lemma assumes $in\text{-}equiv\ A\ A'$ **shows** [*simp*]: $perm\ (In\ A)\ (In\ A')$

lemma *Subst-cancel-left-type*: $set\ x \cap set\ z = \{\} \Longrightarrow TVs\ x = TVs\ y \Longrightarrow Subst\ (x\ @\ z)\ (y\ @\ z)\ w = Subst\ x\ y\ w$

lemma *diff-eq-set-right*: $set\ y = set\ z \Longrightarrow (x \ominus y) = (x \ominus z)$

lemma [*simp*]: $set\ (y \ominus x) \cap set\ x = \{\}$

lemma *in-equiv-Comp*: $io\text{-}diagram\ A' \Longrightarrow io\text{-}diagram\ B' \Longrightarrow in\text{-}equiv\ A\ A' \Longrightarrow in\text{-}equiv\ B\ B' \Longrightarrow in\text{-}equiv\ (A\ ;;\ B)\ (A'\ ;;\ B')$

lemma $io\text{-}diagram\ A' \Longrightarrow io\text{-}diagram\ B' \Longrightarrow in\text{-}equiv\ A\ A' \Longrightarrow in\text{-}equiv\ B\ B' \Longrightarrow in\text{-}equiv\ (CompA\ A\ \ B)\ (CompA\ A'\ B')$

thm *in-equiv-tran*

thm *in-equiv-CompA-Parallel-c*

lemma *comp-parallel-distrib-a*: $TO\ A = TI\ B \Longrightarrow (A\ oo\ B) \parallel C = (A \parallel (ID\ (TI\ C)))\ oo\ (B \parallel C)$

lemma *comp-parallel-distrib-b*: $TO\ A = TI\ B \Longrightarrow C \parallel (A\ oo\ B) = ((ID\ (TI\ C)) \parallel A)\ oo\ (C \parallel B)$

thm *switch-comp-subst*

lemma *CCC-d*: $distinct\ x \Longrightarrow distinct\ y' \Longrightarrow set\ y \subseteq set\ x \Longrightarrow set\ z \subseteq set\ x \Longrightarrow set\ u \subseteq set\ y' \Longrightarrow TVs\ y = TVs\ y' \Longrightarrow$
$TVs\ z = ts \Longrightarrow [x \rightsquigarrow y\ @\ z]\ oo\ [y' \rightsquigarrow u] \parallel (ID\ ts) = [x \rightsquigarrow Subst\ y'\ y\ u\ @\ z]$

lemma *CCC-e*: $distinct\ x \Longrightarrow distinct\ y' \Longrightarrow set\ y \subseteq set\ x \Longrightarrow set\ z \subseteq set\ x \Longrightarrow set\ u \subseteq set\ y' \Longrightarrow TVs\ y = TVs\ y' \Longrightarrow$

$TVs\ z = ts \Longrightarrow [x \rightsquigarrow z\ @\ y]\ oo\ (ID\ ts) \parallel [y' \rightsquigarrow u] = [x \rightsquigarrow z\ @\ Subst\ y'\ y\ u]$

lemma *CCC-a*: $distinct\ x \Longrightarrow distinct\ y \Longrightarrow set\ y \subseteq set\ x \Longrightarrow set\ z \subseteq set\ x \Longrightarrow set\ u \subseteq set\ y \Longrightarrow TVs\ z = ts \Longrightarrow [x \rightsquigarrow y\ @\ z]\ oo\ [y \rightsquigarrow u] \parallel (ID\ ts) = [x \rightsquigarrow u\ @\ z]$

lemma *CCC-b*: $distinct\ x \Longrightarrow distinct\ z \Longrightarrow set\ y \subseteq set\ x \Longrightarrow set\ z \subseteq set\ x \Longrightarrow set\ u \subseteq set\ z \Longrightarrow TVs\ y = ts \Longrightarrow [x \rightsquigarrow y\ @\ z]\ oo\ (ID\ ts) \parallel [z \rightsquigarrow u] = [x \rightsquigarrow y\ @\ u]$

thm *par-switch-eq-dist*

lemma *in-equiv-CompA-Parallel-b*: $length\ (Out\ A) = 1 \Longrightarrow io\text{-}diagram\ A \Longrightarrow io\text{-}diagram\ B \Longrightarrow io\text{-}diagram\ C \Longrightarrow out\ A \in set\ (In\ B) \Longrightarrow out\ A \notin set\ (In\ C) \Longrightarrow in\text{-}equiv\ (CompA\ A\ B\ |||\ CompA\ A\ C)\ (CompA\ A\ (B\ |||\ C))$

lemma *in-equiv-CompA-Parallel-d*: $length\ (Out\ A) = 1 \Longrightarrow io\text{-}diagram\ A \Longrightarrow io\text{-}diagram\ B \Longrightarrow io\text{-}diagram\ C \Longrightarrow out\ A \notin set\ (In\ B) \Longrightarrow out\ A \notin set\ (In\ C) \Longrightarrow in\text{-}equiv\ (CompA\ A\ B\ |||\ CompA\ A\ C)\ (CompA\ A\ (B\ |||\ C))$

lemma *in-equiv-CompA-Parallel*: $deterministic\ (Trs\ A) \Longrightarrow length\ (Out\ A) = 1 \Longrightarrow io\text{-}diagram\ A \Longrightarrow io\text{-}diagram\ B \Longrightarrow io\text{-}diagram\ C \Longrightarrow in\text{-}equiv\ ((A \rhd B)\ |||\ (A \rhd C))\ (A \rhd (B\ |||\ C))$

lemma *fb-less-step-compA*: $deterministic\ (Trs\ A) \Longrightarrow length\ (Out\ A) = 1 \Longrightarrow io\text{-}diagram\ A \Longrightarrow Type\text{-}OK\ As \Longrightarrow in\text{-}equiv\ (Parallel\text{-}list\ (fb\text{-}less\text{-}step\ A\ As))\ (CompA\ A\ (Parallel\text{-}list\ As))$

lemma *switch-eq-Subst*: $distinct\ x \Longrightarrow distinct\ u \Longrightarrow set\ y \subseteq set\ x \Longrightarrow set\ v \subseteq set\ u \Longrightarrow TVs\ x = TVs\ u \Longrightarrow Subst\ x\ u\ y = v \Longrightarrow [x \rightsquigarrow y] = [u \rightsquigarrow v]$

lemma [*simp*]: $set\ y \subseteq set\ y1 \Longrightarrow distinct\ x1 \Longrightarrow TVs\ x1 = TVs\ y1 \Longrightarrow Subst\ x1\ y1\ (Subst\ y1\ x1\ y) = y$

lemma [*simp*]: $set\ z \subseteq set\ x \Longrightarrow TVs\ x\ = TVs\ y \Longrightarrow set\ (Subst\ x\ y\ z) \subseteq set\ y$

thm *distinct-Subst*

lemma *distinct-Subst-aa*: $\bigwedge\ y\ .$
$distinct\ y \Longrightarrow length\ x = length\ y \Longrightarrow a \notin set\ y \Longrightarrow set\ z \cap (set\ y - set\ x) = \{\} \Longrightarrow a \neq aa$
$\Longrightarrow a \notin set\ z \Longrightarrow aa \notin set\ z \Longrightarrow distinct\ z\ \Longrightarrow aa \in set\ x$
$\Longrightarrow subst\ x\ y\ a \neq subst\ x\ y\ aa$

lemma *distinct-Subst-ba*: $distinct\ y \Longrightarrow length\ x = length\ y \Longrightarrow set\ z \cap (set\ y - set\ x) = \{\}$
$\Longrightarrow a \notin set\ z \Longrightarrow distinct\ z\ \Longrightarrow a \notin set\ y \Longrightarrow subst\ x\ y\ a \notin set\ (Subst\ x\ y\ z)$

lemma *distinct-Subst-ca*: $distinct\ y \Longrightarrow length\ x = length\ y \Longrightarrow set\ z \cap (set\ y - set\ x) = \{\}$
$\Longrightarrow a \notin set\ z \Longrightarrow distinct\ z \Longrightarrow a \in set\ x \Longrightarrow subst\ x\ y\ a \notin set\ (Subst\ x\ y\ z)$

lemma [*simp*]: $set\ z \cap (set\ y\ -\ set\ x) = \{\} \Longrightarrow\ distinct\ y \Longrightarrow distinct\ z \Longrightarrow length\ x = length\ y$
$\Longrightarrow distinct\ (Subst\ x\ y\ z)$

lemma *deterministic-Comp*: $io\text{-}diagram\ A \Longrightarrow io\text{-}diagram\ B \Longrightarrow deterministic\ (Trs\ A) \Longrightarrow deterministic\ (Trs\ B)$
$\Longrightarrow deterministic\ (Trs\ (A\ ;;\ B))$

lemma *deterministic-CompA*: $io\text{-}diagram\ A \Longrightarrow io\text{-}diagram\ B \Longrightarrow deterministic\ (Trs\ A) \Longrightarrow deterministic\ (Trs\ B)$
$\Longrightarrow deterministic\ (Trs\ (A \rhd B))$

lemma *parallel-list-empty*[*simp*]: $parallel\text{-}list\ [] = ID\ []$

lemma *parallel-list-append*: $parallel\text{-}list\ (As\ @\ Bs) = parallel\text{-}list\ As\ \|\ parallel\text{-}list\ Bs$

lemma *par-swap-aux*: $distinct\ p \Longrightarrow distinct\ (v\ @\ u\ @\ w) \Longrightarrow$

$TI\ A = TVs\ x \Longrightarrow TI\ B = TVs\ y \Longrightarrow TI\ C = TVs\ z \Longrightarrow$
$TO\ A = TVs\ u \Longrightarrow TO\ B = TVs\ v \Longrightarrow TO\ C = TVs\ w \Longrightarrow$
$set\ x \subseteq set\ p \Longrightarrow set\ y \subseteq set\ p \Longrightarrow set\ z \subseteq set\ p \Longrightarrow set\ q \subseteq set\ (u\ @\ v$

$@\ w) \Longrightarrow$
$[p \rightsquigarrow x\ @\ y\ @\ z]\ oo\ (A \parallel B \parallel C)\ oo\ [u\ @\ v\ @\ w \rightsquigarrow q] = [p \rightsquigarrow y\ @\ x\ @\ z]\ oo\ (B \parallel A \parallel C)\ oo\ [v\ @\ u\ @\ w \rightsquigarrow q]$

lemma *Type-OK-distinct*: *Type-OK As* $\Longrightarrow$ *distinct As*

lemma *TI-parallel-list-a*: *TI* (*parallel-list As*) = *concat* (*map TI As*)

lemma *fb-CompA-aux*: *Type-OK As* $\Longrightarrow A \in$ *set As* $\Longrightarrow$ *out A* = *a* $\Longrightarrow a \notin$ *set* (*In A*) $\Longrightarrow$
InAs = *In* (*Parallel-list As*) $\Longrightarrow$ *OutAs* = *Out* (*Parallel-list As*) $\Longrightarrow$ *perm* (*a* # *y*) *InAs* $\Longrightarrow$ *perm* (*a* # *z*) *OutAs* $\Longrightarrow$
InAs' = *In* (*Parallel-list* (*As* $\ominus$ [*A*])) $\Longrightarrow$
fb ([*a* # *y* $\rightsquigarrow$ *concat* (*map In As*)] *oo parallel-list* (*map Trs As*) *oo* [*OutAs* $\rightsquigarrow$ *a* # *z*]) =
[*y* $\rightsquigarrow$ *In A* @ (*InAs'* $\ominus$ [*a*])]
oo (*Trs A* $\parallel$ [(*InAs'* $\ominus$ [*a*]) $\rightsquigarrow$ (*InAs'* $\ominus$ [*a*])])
oo [*a* # (*InAs'* $\ominus$ [*a*]) $\rightsquigarrow$ *InAs'*] *oo Trs* (*Parallel-list* (*As* $\ominus$ [*A*]))
oo [*OutAs* $\ominus$ [*a*] $\rightsquigarrow$ *z*] (**is** -$\Longrightarrow$ - $\Longrightarrow$ - $\Longrightarrow$ - $\Longrightarrow$ - $\Longrightarrow$ - $\Longrightarrow$ - $\Longrightarrow$ - $\Longrightarrow$ - $\Longrightarrow$ *fb ?Ta* = *?Tb*)

lemma [*simp*]: *perm* (*a* # *x*) (*a* # *y*) = *perm x y*

lemma *fb-CompA*: *Type-OK As* $\Longrightarrow A \in$ *set As* $\Longrightarrow$ *out A* = *a* $\Longrightarrow a \notin$ *set* (*In A*) $\Longrightarrow C = A \rhd$ (*Parallel-list* (*As* $\ominus$ [*A*])) $\Longrightarrow$
OutAs = *Out* (*Parallel-list As*) $\Longrightarrow$ *perm y* (*In C*) $\Longrightarrow$ *perm z* (*Out C*) $\Longrightarrow B \in$ *set As* $- \{A\} \Longrightarrow a \in$ *set* (*In B*) $\Longrightarrow$
fb ([*a* # *y* $\rightsquigarrow$ *concat* (*map In As*)] *oo parallel-list* (*map Trs As*) *oo* [*OutAs* $\rightsquigarrow$ *a* # *z*]) = [*y* $\rightsquigarrow$ *In C*] *oo Trs C oo* [*Out C* $\rightsquigarrow$ *z*]

definition *Deterministic As* = ($\forall$ *A* $\in$ *set As* . *deterministic* (*Trs A*))

lemma *Deterministic-fb-out-less-step*: *Type-OK As* $\Longrightarrow A \in$ *set As* $\Longrightarrow a$ = *out A* $\Longrightarrow$ *Deterministic As* $\Longrightarrow$ *Deterministic* (*fb-out-less-step a As*)

lemma *in-equiv-fb-fb-less-step-TO-CHECK*: *loop-free As* $\Longrightarrow$ *Type-OK As* $\Longrightarrow$ *Deterministic As* $\Longrightarrow$
VarFB (*Parallel-list As*) = *a* # *L* $\Longrightarrow$ *Bs* = *fb-out-less-step a As*
$\Longrightarrow$ *in-equiv* (*FB* (*Parallel-list As*)) (*FB* (*Parallel-list Bs*))

lemma *io-diagram-FB-Parallel-list*: *Type-OK As* $\Longrightarrow$ *io-diagram* (*FB* (*Parallel-list As*))

lemma [*simp*]: *io-diagram* $A \Longrightarrow (|In = In\ A,\ Out = Out\ A,\ Trs = \ Trs\ A|) = A$

thm *loop-free-def*

lemma *io-rel-compA*: *length* $(Out\ A) = 1 \Longrightarrow$ *io-rel* $(CompA\ A\ B) \subseteq$ *io-rel* $B \cup$ (*io-rel* B O *io-rel* A)

theorem *loop-free-fb-out-less-step*: *loop-free* $As \Longrightarrow$ *Type-OK* $As \Longrightarrow A \in$ *set* $As \Longrightarrow$ *out* $A = a \Longrightarrow$ *loop-free* (*fb-out-less-step* a As)

theorem *in-equiv-FB-fb-less-delete*: $\bigwedge$ As . *Deterministic* $As \Longrightarrow$ *loop-free* $As \Longrightarrow$ *Type-OK* $As \Longrightarrow$ *VarFB* (*Parallel-list* As) $= L \Longrightarrow$
in-equiv (*FB* (*Parallel-list* As)) (*Parallel-list* (*fb-less* L As)) $\wedge$ *io-diagram* (*Parallel-list* (*fb-less* L As))

lemmas [*simp*] = *diff-emptyset*

lemma [*simp*]: $\bigwedge$ x . *distinct* $x \Longrightarrow$ *distinct* $y \Longrightarrow$ *perm* $(((y \otimes x) @ (x \ominus y \otimes x)))\ x$

lemma [*simp*]: *io-diagram* $X \Longrightarrow$ *perm* (*VarFB* X @ (*In* $X \ominus$ *VarFB* X)) (*In* X)

lemma *Type-OK-diff* [*simp*]: *Type-OK* $As \Longrightarrow$ *Type-OK* $(As \ominus Bs)$

lemma *internal-fb-out-less-step*:
  assumes [*simp*]: *loop-free* As
    assumes [*simp*]: *Type-OK* As
    and [*simp*]: $a \in$ *internal* As
  shows *internal* (*fb-out-less-step* a As) = *internal* $As - \{a\}$

end

context *BaseOperationFeedbacklessVars*
begin

lemma [*simp*]: *Type-OK* $As \Longrightarrow a \in$ *internal* $As \Longrightarrow$ *out* (*get-comp-out* a As) $= a$

lemma *internal-Type-OK-simp*: *Type-OK* $As \Longrightarrow$ *internal* $As = \{a\ .\ (\exists\ A \in$ *set* As . *out* $A = a \wedge (\exists\ B \in$ *set* $As.\ a \in$ *set* $(In\ B)))\}$

thm *Type-OK-def*

lemma *Type-OK-fb-less*: $\bigwedge$ *As* . *Type-OK As* $\Longrightarrow$ *loop-free As* $\Longrightarrow$ *distinct x* $\Longrightarrow$ *set x* $\subseteq$ *internal As* $\Longrightarrow$ *Type-OK* (*fb-less x As*)

lemma *fb-Parallel-list-fb-out-less-step*:
assumes [*simp*]: *Type-OK As*
and *Deterministic As*
and *loop-free As*
and *internal*: $a \in$ *internal As*
and X: X = *Parallel-list As*
and Y: Y = (*Parallel-list* (*fb-out-less-step a As*))
and [*simp*]: *perm y* (*In Y*)
and [*simp*]: *perm z* (*Out Y*)
shows *fb* ([*a* # *y* $\rightsquigarrow$ *In X*] *oo Trs X oo* [*Out X* $\rightsquigarrow$ *a* # *z*]) = [*y* $\rightsquigarrow$ *In Y*] *oo Trs Y oo* [*Out Y* $\rightsquigarrow$ *z*] **and** *perm* (*a* # *In Y*) (*In X*)

lemma *internal-In-Parallel-list*: $a \in$ *internal As* $\Longrightarrow$ $a \in$ *set* (*In* (*Parallel-list As*))

lemma *internal-Out-Parallel-list*: $a \in$ *internal As* $\Longrightarrow$ $a \in$ *set* (*Out* (*Parallel-list As*))

theorem *fb-power-internal-fb-less*: $\bigwedge$ *As X Y* . *Deterministic As* $\Longrightarrow$ *loop-free As* $\Longrightarrow$ *Type-OK As* $\Longrightarrow$ *set L* $\subseteq$ *internal As*
$\Longrightarrow$ *distinct L* $\Longrightarrow$
X = (*Parallel-list As*) $\Longrightarrow$ Y = *Parallel-list* (*fb-less L As*) $\Longrightarrow$
(*fb* ^^ *length* (*L*)) ([*L* @ (*In X* $\ominus$ *L*) $\rightsquigarrow$ *In X*] *oo Trs X oo* [*Out X* $\rightsquigarrow$ *L* @ (*Out X* $\ominus$ *L*)]) = [*In X* $\ominus$ *L* $\rightsquigarrow$ *In Y*] *oo Trs Y*
$\wedge$ *perm* (*In X* $\ominus$ *L*) (*In Y*)

thm *fb-power-internal-fb-less*

theorem *FB-fb-less*:
assumes [*simp*]: *Deterministic As*
and [*simp*]: *loop-free As*
and [*simp*]: *Type-OK As*
and [*simp*]: *perm* (*VarFB X*) *L*
and X: X = (*Parallel-list As*)
and Y: Y = *Parallel-list* (*fb-less L As*)
shows (*fb* ^^ *length* (*L*)) ([*L* @ *InFB X* $\rightsquigarrow$ *In X*] *oo Trs X oo* [*Out X* $\rightsquigarrow$ *L* @ *OutFB X*]) = [*InFB X* $\rightsquigarrow$ *In Y*] *oo Trs Y*
and *B*: *perm* (*InFB X*) (*In Y*)

definition *fb-perm-eq* $A = (\forall\ x.\ perm\ x\ (VarFB\ A) \longrightarrow$
$(fb\ \hat{}\hat{}\ length\ (VarFB\ A))\ ([VarFB\ A\ @\ InFB\ A \rightsquigarrow In\ A]\ oo\ Trs\ A\ oo\ [Out\ A \rightsquigarrow VarFB\ A\ @\ OutFB\ A]) =$
$(fb\ \hat{}\hat{}\ length\ (VarFB\ A))\ ([x\ @\ InFB\ A \rightsquigarrow In\ A]\ oo\ Trs\ A\ oo\ [Out\ A \rightsquigarrow x\ @\ OutFB\ A]))$

lemma *fb-perm-eq-simp*: *fb-perm-eq* $A = (\forall\ x.\ perm\ x\ (VarFB\ A) \longrightarrow$
$Trs\ (FB\ A) = (fb\ \hat{}\hat{}\ length\ (VarFB\ A))\ ([x\ @\ InFB\ A \rightsquigarrow In\ A]\ oo\ Trs\ A\ oo\ [Out\ A \rightsquigarrow x\ @\ OutFB\ A]))$

lemma *in-equiv-in-out-equiv*: *io-diagram* $B \Longrightarrow$ *in-equiv* $A\ B \Longrightarrow$ *in-out-equiv* $A\ B$

lemma [*simp*]: $distinct\ (concat\ (map\ f\ As)) \Longrightarrow distinct\ (concat\ (map\ f\ (As \ominus [A])))$

lemma *set-op-list-addvars*: $set\ (op\text{-}list\ []\ (\oplus)\ x) = (\bigcup\ a \in set\ x\ .\ set\ a)$

end

context *BaseOperationFeedbacklessVars*

begin

lemma [*simp*]: $set\ (Out\ A) \subseteq set\ (In\ B) \Longrightarrow Out\ ((A\ ;;\ B)) = Out\ B$

lemma [*simp*]: $set\ (Out\ A) \subseteq set\ (In\ B) \Longrightarrow out\ ((A\ ;;\ B)) = out\ B$

lemma *switch-par-comp3*:
assumes [*simp*]: *distinct* x **and**
[*simp*]: *distinct* y
and [*simp*]: *distinct* z
and [*simp*]: *distinct* u
and [*simp*]: $set\ y \subseteq set\ x$
and [*simp*]: $set\ z \subseteq set\ x$
and [*simp*]: $set\ u \subseteq set\ x$
and [*simp*]: $set\ y' \subseteq set\ y$
and [*simp*]: $set\ z' \subseteq set\ z$
and [*simp*]: $set\ u' \subseteq set\ u$
shows $[x \rightsquigarrow y\ @\ z\ @\ u]\ oo\ [y \rightsquigarrow y'] \parallel [z \rightsquigarrow z'] \parallel [u \rightsquigarrow u'] = [x \rightsquigarrow y'\ @\ z'\ @\ u']$

lemma *switch-par-comp-Subst3*:
assumes [*simp*]: *distinct* x **and** [*simp*]: *distinct* y' **and** [*simp*]: *distinct* z' **and** [*simp*]: *distinct* t'
and [*simp*]: *set* $y \subseteq$ *set* x **and** [*simp*]: *set* $z \subseteq$ *set* x **and** [*simp*]: *set* $t \subseteq$ *set* x
and [*simp*]: *set* $u \subseteq$ *set* y' **and** [*simp*]: *set* $v \subseteq$ *set* z' **and** [*simp*]: *set* $w \subseteq$ *set* t'
and [*simp*]: *TVs* y = *TVs* y' **and** [*simp*]: *TVs* z = *TVs* z' **and** [*simp*]: *TVs* t = *TVs* t'

shows $[x \rightsquigarrow y \mathbin{@} z \mathbin{@} t]$ *oo* $[y' \rightsquigarrow u] \parallel [z' \rightsquigarrow v] \parallel [t' \rightsquigarrow w] = [x \rightsquigarrow \mathit{Subst}\ y'\ y\ u \mathbin{@} \mathit{Subst}\ z'\ z\ v \mathbin{@} \mathit{Subst}\ t'\ t\ w]$

lemma *Comp-assoc-single*: *length* (*Out A*) = *1* $\Longrightarrow$ *length* (*Out B*) = *1* $\Longrightarrow$ *out A* $\neq$ *out B* $\Longrightarrow$ *io-diagram A*
$\Longrightarrow$ *io-diagram B* $\Longrightarrow$ *io-diagram C* $\Longrightarrow$ *out B* $\notin$ *set* (*In A*) $\Longrightarrow$
deterministic (*Trs A*) $\Longrightarrow$
out A $\in$ *set* (*In B*) $\Longrightarrow$ *out A* $\in$ *set* (*In C*) $\Longrightarrow$ *out B* $\in$ *set* (*In C*) $\Longrightarrow$ (*A* ;; (*B* ;; *C*)) = (*A* ;; *B* ;; (*A* ;; *C*))

lemma *Comp-commute-aux*:
assumes [*simp*]: *length* (*Out A*) = *1*
and [*simp*]: *length* (*Out B*) = *1*
and [*simp*]: *io-diagram A*
and [*simp*]: *io-diagram B*
and [*simp*]: *io-diagram C*
and [*simp*]: *out B* $\notin$ *set* (*In A*)
and [*simp*]: *out A* $\notin$ *set* (*In B*)
and [*simp*]: *out A* $\in$ *set* (*In C*)
and [*simp*]: *out B* $\in$ *set* (*In C*)
and *Diff*: *out A* $\neq$ *out B*

shows *Trs* (*A* ;; (*B* ;; *C*)) =
[*In A* $\oplus$ *In B* $\oplus$ (*In C* $\ominus$ [*out A*] $\ominus$ [*out B*]) $\rightsquigarrow$ *In A* @ *In B* @ (*In C* $\ominus$ [*out A*] $\ominus$ [*out B*])]
oo Trs A $\parallel$ *Trs B* $\parallel$ [*In C* $\ominus$ [*out A*] $\ominus$ [*out B*] $\rightsquigarrow$ *In C* $\ominus$ [*out A*] $\ominus$ [*out B*]]
oo [*out A* # *out B* # (*In C* $\ominus$ [*out A*] $\ominus$ [*out B*]) $\rightsquigarrow$ *In C*]
oo Trs C
and *In* (*A* ;; (*B* ;; *C*)) = *In A* $\oplus$ *In B* $\oplus$ (*In C* $\ominus$ [*out A*] $\ominus$ [*out B*])
and *Out* (*A* ;; (*B* ;; *C*)) = *Out C*

lemma *Comp-commute*:
assumes [*simp*]: *length* (*Out A*) = *1*
and [*simp*]: *length* (*Out B*) = *1*
and [*simp*]: *io-diagram A*
and [*simp*]: *io-diagram B*
and [*simp*]: *io-diagram C*

and [*simp*]: *out B* $\notin$ *set* (*In A*)
and [*simp*]: *out A* $\notin$ *set* (*In B*)
and [*simp*]: *out A* $\in$ *set* (*In C*)
and [*simp*]: *out B* $\in$ *set* (*In C*)
and *Diff*: *out A* $\neq$ *out B*
shows *in-equiv* (*A* ;; (*B* ;; *C*)) (*B* ;; (*A* ;; *C*))

lemma *CompA-commute-aux-a*: *io-diagram A* $\Longrightarrow$ *io-diagram B* $\Longrightarrow$ *io-diagram C* $\Longrightarrow$ *length* (*Out A*) = *1* $\Longrightarrow$ *length* (*Out B*) = *1*
$\Longrightarrow$ *out A* $\notin$ *set* (*Out C*) $\Longrightarrow$ *out B* $\notin$ *set* (*Out C*)
$\Longrightarrow$ *out A* $\neq$ *out B* $\Longrightarrow$ *out A* $\in$ *set* (*In B*) $\Longrightarrow$ *out B* $\notin$ *set* (*In A*)
$\Longrightarrow$ *deterministic* (*Trs A*)
$\Longrightarrow$ (*CompA* (*CompA B A*) (*CompA B C*)) = (*CompA* (*CompA A B*) (*CompA A C*))

lemma *CompA-commute-aux-b*: *io-diagram A* $\Longrightarrow$ *io-diagram B* $\Longrightarrow$ *io-diagram C* $\Longrightarrow$ *length* (*Out A*) = *1* $\Longrightarrow$ *length* (*Out B*) = *1*
$\Longrightarrow$ *out A* $\notin$ *set* (*Out C*) $\Longrightarrow$ *out B* $\notin$ *set* (*Out C*)
$\Longrightarrow$ *out A* $\neq$ *out B* $\Longrightarrow$ *out A* $\notin$ *set* (*In B*) $\Longrightarrow$ *out B* $\notin$ *set* (*In A*)
$\Longrightarrow$ *in-equiv* (*CompA* (*CompA B A*) (*CompA B C*)) (*CompA* (*CompA A B*) (*CompA A C*))

fun *In-Equiv* :: ((*'var*, *'a*) *Dgr*) *list* $\Rightarrow$ ((*'var*, *'a*) *Dgr*) *list* $\Rightarrow$ *bool* **where**
In-Equiv [] [] = *True* |
In-Equiv (*A* # *As*) (*B*# *Bs*) = (*in-equiv A B* $\wedge$ *In-Equiv As Bs*) |
In-Equiv - - = *False*

thm *internal-def*

thm *fb-out-less-step-def*
thm *fb-less-step-def*

thm *CompA-commute-aux-b*
thm *CompA-commute-aux-a*

lemma *CompA-commute*:
assumes [*simp*]: *io-diagram A*
and [*simp*]: *io-diagram B*
and [*simp*]: *io-diagram C*
and [*simp*]: *length* (*Out A*) = *1*
and [*simp*]: *length* (*Out B*) = *1*
and [*simp*]: *out A* $\notin$ *set* (*Out C*)
and [*simp*]: *out B* $\notin$ *set* (*Out C*)
and [*simp*]: *out A* $\neq$ *out B*
and [*simp*]: *deterministic* (*Trs A*)
and [*simp*]: *deterministic* (*Trs B*)
and *A*: (*out A* $\in$ *set* (*In B*) $\Longrightarrow$ *out B* $\notin$ *set* (*In A*))

shows *in-equiv* (*CompA* (*CompA B A*) (*CompA B C*)) (*CompA* (*CompA A B*) (*CompA A C*))

lemma *In-Equiv-CompA-twice*: ($\bigwedge$ *C* . *C* $\in$ *set As* $\Longrightarrow$ *io-diagram C* $\wedge$ *out A* $\notin$ *set* (*Out C*) $\wedge$ *out B* $\notin$ *set* (*Out C*)) $\Longrightarrow$ *io-diagram A* $\Longrightarrow$ *io-diagram B*
$\Longrightarrow$ *length* (*Out A*) = *1* $\Longrightarrow$ *length* (*Out B*) = *1* $\Longrightarrow$ *out A* $\neq$ *out B*
$\Longrightarrow$ *deterministic* (*Trs A*) $\Longrightarrow$ *deterministic* (*Trs B*)
$\Longrightarrow$ (*out A* $\in$ *set* (*In B*) $\Longrightarrow$ *out B* $\notin$ *set* (*In A*))
$\Longrightarrow$ *In-Equiv* (*map* (*CompA* (*CompA B A*)) (*map* (*CompA B*) *As*)) (*map* (*CompA* (*CompA A B*)) (*map* (*CompA A*) *As*))

thm *Type-OK-def*
thm *Deterministic-def*
thm *internal-def*
thm *fb-out-less-step-def*

thm *mem-get-other-out*

thm *mem-get-comp-out*

thm *comp-out-in*

lemma *map-diff*: ($\bigwedge$ *b* . *b* $\in$ *set x* $\Longrightarrow$ *b* $\neq$ *a* $\Longrightarrow$ *f b* $\neq$ *f a*) $\Longrightarrow$ *map f x* $\ominus$ [*f a*] = *map f* (*x* $\ominus$ [*a*])

lemma *In-Equiv-fb-out-less-step-commute*: *Type-OK As* $\Longrightarrow$ *Deterministic As* $\Longrightarrow$ *x* $\in$ *internal As* $\Longrightarrow$ *y* $\in$ *internal As* $\Longrightarrow$ *x* $\neq$ *y* $\Longrightarrow$ *loop-free As*
$\Longrightarrow$ *In-Equiv* (*fb-out-less-step x* (*fb-out-less-step y As*)) (*fb-out-less-step y* (*fb-out-less-step x As*))

lemma [*simp*]: *Type-OK As* $\Longrightarrow$ *In-Equiv As As*

lemma *fb-less-append*: $\bigwedge$ *As* . *fb-less* (*x* @ *y*) *As* = *fb-less y* (*fb-less x As*)

thm *in-equiv-tran*

lemma *In-Equiv-trans*: $\bigwedge$ *Bs Cs* . *Type-OK Cs* $\Longrightarrow$ *In-Equiv As Bs* $\Longrightarrow$ *In-Equiv Bs Cs* $\Longrightarrow$ *In-Equiv As Cs*

lemma *In-Equiv-exists*: $\bigwedge$ *Bs* . *In-Equiv As Bs* $\Longrightarrow$ *A* $\in$ *set As* $\Longrightarrow$ $\exists$ *B* $\in$ *set Bs* . *in-equiv A B*

lemma *In-Equiv-Type-OK*: $\bigwedge Bs\ .\ \textit{Type-OK Bs} \Longrightarrow \textit{In-Equiv As Bs} \Longrightarrow \textit{Type-OK}$ As

lemma *In-Equiv-internal-aux*: $\textit{Type-OK Bs} \Longrightarrow \textit{In-Equiv As Bs} \Longrightarrow \textit{internal As}$ $\subseteq \textit{internal Bs}$

lemma *In-Equiv-sym*: $\bigwedge\ Bs\ .\ \textit{Type-OK Bs} \Longrightarrow \textit{In-Equiv As Bs} \Longrightarrow \textit{In-Equiv Bs}$ As

lemma *In-Equiv-internal*: $\textit{Type-OK Bs} \Longrightarrow \textit{In-Equiv As Bs} \Longrightarrow \textit{internal As} =$ $\textit{internal Bs}$

lemma *in-equiv-CompA*: $\textit{in-equiv } A\ A' \Longrightarrow \textit{in-equiv } B\ B' \Longrightarrow \textit{io-diagram } A' \Longrightarrow$ $\textit{io-diagram } B' \Longrightarrow \textit{in-equiv } (\textit{CompA } A\ B)\ (\textit{CompA } A'\ B')$

lemma *In-Equiv-fb-less-step-cong*: $\bigwedge\ Bs\ .\ \textit{Type-OK Bs} \Longrightarrow \textit{in-equiv } A\ B \Longrightarrow$ $\textit{io-diagram } B \Longrightarrow \textit{In-Equiv As Bs}$
$\Longrightarrow \textit{In-Equiv } (\textit{fb-less-step } A\ As)\ (\textit{fb-less-step } B\ Bs)$

lemma *In-Equiv-append*: $\bigwedge\ As'\ .\ \textit{In-Equiv As As}' \Longrightarrow \textit{In-Equiv Bs Bs}' \Longrightarrow$ $\textit{In-Equiv } (As\ @\ Bs)\ (As'\ @\ Bs')$

lemma *In-Equiv-split*: $\bigwedge\ Bs\ .\ \textit{In-Equiv As Bs} \Longrightarrow A \in \textit{set As}$
$\Longrightarrow \exists\ B\ As'\ As''\ Bs'\ Bs''\ .\ As = As'\ @\ A\ \#\ As'' \wedge Bs = Bs'\ @\ B\ \#\ Bs'' \wedge$ $\textit{in-equiv } A\ B \wedge \textit{In-Equiv } As'\ Bs' \wedge \textit{In-Equiv } As''\ Bs''$

lemma *In-Equiv-fb-out-less-step-cong*:
assumes [*simp*]: *Type-OK Bs*
and *In-Equiv As Bs*
and *internal*: $a \in \textit{internal As}$
shows $\textit{In-Equiv } (\textit{fb-out-less-step } a\ As)\ (\textit{fb-out-less-step } a\ Bs)$

lemma *In-Equiv-IO-Rel*: $\bigwedge\ Bs\ .\ \textit{In-Equiv As Bs} \Longrightarrow \textit{IO-Rel Bs} = \textit{IO-Rel As}$

lemma *In-Equiv-loop-free*: $\textit{In-Equiv As Bs} \Longrightarrow \textit{loop-free Bs} \Longrightarrow \textit{loop-free As}$

lemma *loop-free-fb-out-less-step-internal*:
assumes [*simp*]: *loop-free As*
and [*simp*]: *Type-OK As*
and $a \in \textit{internal As}$
shows $\textit{loop-free } (\textit{fb-out-less-step } a\ As)$

lemma *loop-free-fb-less-internal*:
$\bigwedge As\ .\ \textit{loop-free As} \Longrightarrow \textit{Type-OK As} \Longrightarrow \textit{set } x \subseteq \textit{internal As} \Longrightarrow \textit{distinct } x \Longrightarrow$ $\textit{loop-free } (\textit{fb-less } x\ As)$

lemma *In-Equiv-fb-less-cong*: $\bigwedge$ *As Bs* . *Type-OK Bs* $\Longrightarrow$ *In-Equiv As Bs* $\Longrightarrow$ *set x* $\subseteq$ *internal As* $\Longrightarrow$ *distinct x* $\Longrightarrow$ *loop-free Bs* $\Longrightarrow$ *In-Equiv* (*fb-less x As*) (*fb-less x Bs*)

thm *Type-OK-fb-out-less-step-new*

thm *Type-OK-fb-less*

lemma *Type-OK-fb-less-delete*: $\bigwedge$ *As* . *Type-OK As* $\Longrightarrow$ *set x* $\subseteq$ *internal As* $\Longrightarrow$ *distinct x* $\Longrightarrow$ *loop-free As* $\Longrightarrow$ *Type-OK* (*fb-less x As*)

thm *Deterministic-fb-out-less-step*

thm *internal-fb-out-less-step*

lemma *internal-fb-less*:
$\bigwedge$ *As* . *loop-free As* $\Longrightarrow$ *Type-OK As* $\Longrightarrow$ *set x* $\subseteq$ *internal As* $\Longrightarrow$ *distinct x* $\Longrightarrow$ *internal* (*fb-less x As*) = *internal As* − *set x*

thm *Deterministic-fb-out-less-step*

lemma *Deterministic-fb-out-less-step-internal*:
assumes [*simp*]: *Type-OK As*
and *Deterministic As*
and *internal*: $a \in$ *internal As*
shows *Deterministic* (*fb-out-less-step a As*)

lemma *Deterministic-fb-less-internal*: $\bigwedge$ *As* . *Type-OK As* $\Longrightarrow$ *Deterministic As* $\Longrightarrow$ *set x* $\subseteq$ *internal As* $\Longrightarrow$ *distinct x*
$\Longrightarrow$ *loop-free As* $\Longrightarrow$ *Deterministic* (*fb-less x As*)

lemma *In-Equiv-fb-less-Cons*: $\bigwedge$ *As* . *Type-OK As* $\Longrightarrow$ *Deterministic As* $\Longrightarrow$ *loop-free As* $\Longrightarrow$ $a \in$ *internal As*
$\Longrightarrow$ *set x* $\subseteq$ *internal As* $\Longrightarrow$ *distinct* (a # x)
$\Longrightarrow$ *In-Equiv* (*fb-less* (a # x) *As*) (*fb-less* (x @ [a]) *As*)

theorem *In-Equiv-fb-less*: $\bigwedge$ *y As* . *Type-OK As* $\Longrightarrow$ *Deterministic As* $\Longrightarrow$ *loop-free*

$As \Longrightarrow$ *set* $x \subseteq$ *internal* $As \Longrightarrow$ *distinct* $x \Longrightarrow$ *perm* $x\ y$
$\Longrightarrow$ *In-Equiv* (*fb-less* x As) (*fb-less* y As)

lemma [*simp*]: *in-equiv* □ □

lemma *in-equiv-Parallel-list*: $\bigwedge$ *Bs* . *Type-OK Bs* $\Longrightarrow$ *In-Equiv As Bs* $\Longrightarrow$ *in-equiv* (*Parallel-list As*) (*Parallel-list Bs*)

thm *FB-fb-less*

lemma [*simp*]: *io-diagram* $A \Longrightarrow$ *distinct* (*VarFB* A)

lemma [*simp*]: *io-diagram* $A \Longrightarrow$ *distinct* (*InFB* A)

theorem *fb-perm-eq-Parallel-list*:
  assumes [*simp*]: *Type-OK As*
    and [*simp*]: *Deterministic As*
    and [*simp*]: *loop-free As*
    shows *fb-perm-eq* (*Parallel-list As*)

theorem *FeedbackSerial-Feedbackless*: *io-diagram* $A \Longrightarrow$ *io-diagram* $B \Longrightarrow$ *set* (*In* A) $\cap$ *set* (*In* B) = {} — required
$\Longrightarrow$ *set* (*Out* A) $\cap$ *set* (*Out* B) = {} $\Longrightarrow$ *fb-perm-eq* (A ||| B) $\Longrightarrow$ *FB* (A ||| B) = *FB* (*FB* (A) ;; *FB* (B))

declare *io-diagram-distinct* [*simp del*]

lemma *in-out-equiv-FB-less*: *io-diagram* $B \Longrightarrow$ *in-out-equiv* A $B \Longrightarrow$ *fb-perm-eq* $A \Longrightarrow$ *in-out-equiv* (*FB* A) (*FB* B)

lemma [*simp*]: *io-diagram* $A \Longrightarrow$ *distinct* (*OutFB* A)

end

end

8 Refinement Calculus and Monotonic Predicate Transformers

theory *Refinement* **imports** *Main*
begin

In this section we introduce the basics of refinement calculus [**?**]. Part of this theory is a reformulation of some definitions from [**?**], but here they are given for predicates, while [**?**] uses sets.

notation
  bot (⊥) **and**
  top (⊤) **and**
  inf (**infixl** ⊓ *70*)
  and *sup* (**infixl** ⊔ *65*)

8.1 Basic predicate transformers

definition
  demonic :: ('*a* => '*b*::*lattice*) => '*b* => '*a* ⇒ *bool* ([: - :] [*0*] *1000*) **where**
  [:*Q*:] *p s* = (*Q s* ≤ *p*)

definition
  assert::'*a*::*semilattice-inf* => '*a* => '*a* ({. - .} [*0*] *1000*) **where**
  {.*p*.} *q* ≡ *p* ⊓ *q*

definition
  assume::('*a*::*boolean-algebra*) => '*a* => '*a* ([. - .] [*0*] *1000*) **where**
  [.*p*.] *q* ≡ (−*p* ⊔ *q*)

definition
  angelic :: ('*a* ⇒ '*b*::{*semilattice-inf*,*order-bot*}) ⇒ '*b* ⇒ '*a* ⇒ *bool* ({: - :} [*0*] *1000*) **where**
  {:*Q*:} *p s* = (*Q s* ⊓ *p* ≠ ⊥)

syntax
  -assert :: *patterns* => *logic* => *logic* ((*1*{.-.-.}))
translations
  -assert x P == *CONST assert* (*-abs x P*)

syntax
  -demonic :: *patterns* => *patterns* => *logic* => *logic* (([:-⇝-.-:]))
translations
  -demonic x y t == (*CONST demonic* (*-abs x* (*-abs y t*)))

syntax
  -angelic :: *patterns* => *patterns* => *logic* => *logic* (({:- ⇝ -.-:}))
translations
  -angelic x y t == (*CONST angelic* (*-abs x* (*-abs y t*)))

lemma *assert-o-def*: {.*f o g*.} = {.(λ *x* . *f* (*g x*)).}

lemma *demonic-demonic*: [:*r*:] *o* [:*r'*:] = [:*r OO r'*:]

lemma *assert-demonic-comp*: {.*p*.} *o* [:*r*:] *o* {.*p'*.} *o* [:*r'*:] =
  {.*x* . *p x* ∧ (∀ *y* . *r x y* ⟶ *p' y*).} *o* [:*r OO r'*:]

lemma *demonic-assert-comp*: $[:r:]$ o $\{.p.\} = \{.x.(\forall\ y\ .\ r\ x\ y \longrightarrow p\ y).\}$ o $[:r:]$

lemma *assert-assert-comp*: $\{.p::'a::lattice.\}$ o $\{.p'.\} = \{.p \sqcap p'.\}$

lemma *assert-assert-comp-pred*: $\{.p.\}$ o $\{.p'.\} = \{.x\ .\ p\ x \wedge p'\ x.\}$

lemma *demonic-refinement*: $r' \leq r \Longrightarrow [:r:] \leq [:r':]$

definition *inpt* $r\ x = (\exists\ y\ .\ r\ x\ y)$

definition *trs* :: $('a => 'b \Rightarrow bool) => ('b \Rightarrow bool) => 'a \Rightarrow bool$ ({: - :] [0] 1000) **where**
trs $r = \{.\ inpt\ r.\}$ o $[:r:]$

syntax
-*trs* :: *patterns* => *patterns* => *logic* => *logic* (({:-⇝-.-:]))
translations
-*trs* $x\ y\ t$ == (*CONST trs* (-*abs* x (-*abs* y t)))

lemma *assert-demonic-prop*: $\{.p.\}$ o $[:r:] = \{.p.\}$ o $[:(\lambda\ x\ y\ .\ p\ x) \sqcap r:]$

lemma *trs-trs*: (*trs* r) o (*trs* r')
$=$ *trs* $((\lambda\ s\ t.\ (\forall\ s'\ .\ r\ s\ s' \longrightarrow (inpt\ r'\ s'))) \sqcap (r\ OO\ r'))$ (**is** $?S = ?T$)

lemma *prec-inpt-equiv*: $p \leq inpt\ r \Longrightarrow r' = (\lambda\ x\ y\ .\ p\ x \wedge r\ x\ y) \Longrightarrow \{.p.\}$ o $[:r:]$ $= \{:r':]$

lemma *assert-demonic-refinement*: $(\{.p.\}$ o $[:r:] \leq \{.p'.\}$ o $[:r':]) = (p \leq p' \wedge (\forall\ x\ .\ p\ x \longrightarrow r'\ x \leq r\ x))$

lemma *spec-demonic-refinement*: $(\{.p.\}$ o $[:r:] \leq [:r':]) = (\forall\ x\ .\ p\ x \longrightarrow r'\ x \leq r\ x)$

lemma *trs-refinement*: $(trs\ r \leq trs\ r') = ((\forall\ x\ .\ inpt\ r\ x \longrightarrow inpt\ r'\ x) \wedge (\forall\ x\ .\ inpt\ r\ x \longrightarrow r'\ x \leq r\ x))$

lemma *demonic-choice*: $[:r:] \sqcap [:r':] = [:r \sqcup r':]$

lemma *spec-demonic-choice*: $(\{.p.\}$ o $[:r:]) \sqcap (\{.p'.\}$ o $[:r':]) = (\{.p \sqcap p'.\}$ o $[:r \sqcup r':])$

lemma *trs-demonic-choice*: $trs\ r \sqcap trs\ r' = trs\ ((\lambda\ x\ y\ .\ inpt\ r\ x \wedge inpt\ r'\ x) \sqcap (r \sqcup r'))$

lemma *spec-angelic*: $p \sqcap p' = \bot \Longrightarrow (\{.p.\}\ o\ [:r:]) \sqcup (\{.p'.\}\ o\ [:r':])$
$= \{.p \sqcup p'.\}\ o\ [:(\lambda\ x\ y\ .\ p\ x \longrightarrow r\ x\ y) \sqcap ((\lambda\ x\ y\ .\ p'\ x \longrightarrow r'\ x\ y)):]$

8.2 Conjunctive predicate transformers

definition *conjunctive* $(S::'a::complete\text{-}lattice \Rightarrow 'b::complete\text{-}lattice) = (\forall\ Q\ .\ S\ (Inf\ Q) = INFIMUM\ Q\ S)$

definition *sconjunctive* $(S::'a::complete\text{-}lattice \Rightarrow 'b::complete\text{-}lattice) = (\forall\ Q\ .\ (\exists\ x\ .\ x \in Q) \longrightarrow S\ (Inf\ Q) = INFIMUM\ Q\ S)$

lemma *conjunctive-sconjunctive*[*simp*]: *conjunctive* $S \Longrightarrow$ *sconjunctive* S

lemma [*simp*]: *conjunctive* $\top$

lemma *conjuncive-demonic* [*simp*]: *conjunctive* $[:r:]$

lemma *sconjunctive-assert* [*simp*]: *sconjunctive* $\{.p.\}$

lemma *sconjunctive-simp*: $x \in Q \Longrightarrow$ *sconjunctive* $S \Longrightarrow S\ (Inf\ Q) = INFIMUM\ Q\ S$

lemma *sconjunctive-INF-simp*: $x \in X \Longrightarrow$ *sconjunctive* $S \Longrightarrow S\ (INFIMUM\ X\ Q) = INFIMUM\ (Q`X)\ S$

lemma *demonic-comp* [*simp*]: *sconjunctive* $S \Longrightarrow$ *sconjunctive* $S' \Longrightarrow$ *sconjunctive* $(S\ o\ S')$

lemma *conjunctive-INF*[*simp*]:*conjunctive* $S \Longrightarrow S\ (INFIMUM\ X\ Q) = (INFIMUM\ X\ (S\ o\ Q))$

lemma *conjunctive-simp*: *conjunctive* $S \Longrightarrow\ S\ (Inf\ Q) = INFIMUM\ Q\ S$

lemma *conjunctive-monotonic* [*simp*]: *sconjunctive* $S \Longrightarrow$ *mono* S

definition *grd* $S = -\ S\ \bot$

lemma *grd-demonic*: *grd* $[:r:] =$ *inpt* r

lemma $(S::'a::bot \Rightarrow 'b::boolean\text{-}algebra) \le S' \Longrightarrow grd\ S' \le grd\ S$

lemma [*simp*]: *inpt* $(\lambda x\ y.\ p\ x \wedge r\ x\ y) = p \sqcap$ *inpt* r

lemma [*simp*]: $p \le$ *inpt* $r \Longrightarrow p \sqcap$ *inpt* $r = p$

lemma *grd-spec*: *grd* $(\{.p.\}\ o\ [:r:]) = -p \sqcup$ *inpt* r

definition *fail* $S = -(S\ \top)$
definition *term* $S = (S\ \top)$
definition *prec* $S = -\ (\mathit{fail}\ S)$
definition *rel* $S = (\lambda\ x\ y\ .\ \neg\ S\ (\lambda\ z\ .\ y \neq z)\ x)$

lemma *rel-spec*: $\mathit{rel}\ (\{.p.\}\ o\ [:r:])\ x\ y = (p\ x \longrightarrow r\ x\ y)$

lemma *prec-spec*: $\mathit{prec}\ (\{.p.\}\ o\ [:r::'a \Rightarrow 'b \Rightarrow \mathit{bool}:]) = p$

lemma *fail-spec*: $\mathit{fail}\ (\{.p.\}\ o\ [:(r::'a \Rightarrow 'b::\mathit{boolean\text{-}algebra}):]) = -p$

lemma [*simp*]: $\mathit{prec}\ (\{.p.\}\ o\ [:(r::'a \Rightarrow 'b::\mathit{boolean\text{-}algebra}):]) = p$

lemma [*simp*]: $\mathit{prec}\ (T::('a::\mathit{boolean\text{-}algebra} \Rightarrow\ 'b::\mathit{boolean\text{-}algebra})) = \top \Longrightarrow \mathit{prec}\ (S\ o\ T) = \mathit{prec}\ S$

lemma [*simp*]: $\mathit{prec}\ [:r::'a \Rightarrow\ 'b::\mathit{boolean\text{-}algebra}:] = \top$

lemma *prec-rel*: $\{.\ p\ .\} \circ [:\ \lambda x\ y.\ p\ x \wedge r\ x\ y\ :] = \{.p.\}\ o\ [:r:]$

definition *Fail* $= \bot$

lemma *Fail-assert-demonic*: $\mathit{Fail} = \{.\bot.\}\ o\ [:r:]$

lemma *Fail-assert*: $\mathit{Fail} = \{.\bot.\}\ o\ [:\bot:]$

lemma *fail-comp*[*simp*]: $\bot\ o\ S = \bot$

lemma *Fail-fail*: $\mathit{mono}\ (S::'a::\mathit{boolean\text{-}algebra} \Rightarrow\ 'b::\mathit{boolean\text{-}algebra}) \Longrightarrow (S = \mathit{Fail}) = (\mathit{fail}\ S = \top)$

lemma *sconjunctive-spec*: $\mathit{sconjunctive}\ S \Longrightarrow S = \{.\mathit{prec}\ S.\}\ o\ [:\mathit{rel}\ S:]$

definition *non-magic* $S = (S\ \bot = \bot)$

lemma *non-magic-spec*: $\mathit{non\text{-}magic}\ (\{.p.\}\ o\ [:r:]) = (p \leq \mathit{inpt}\ r)$

lemma *sconjunctive-non-magic*: $\mathit{sconjunctive}\ S \Longrightarrow \mathit{non\text{-}magic}\ S = (\mathit{prec}\ S \leq \mathit{inpt}\ (\mathit{rel}\ S))$

definition *implementable* $S = (\mathit{sconjunctive}\ S \wedge \mathit{non\text{-}magic}\ S)$

lemma *implementable-spec*: $\mathit{implementable}\ S \Longrightarrow \exists\ p\ r\ .\ S = \{.p.\}\ o\ [:r:] \wedge p \leq \mathit{inpt}\ r$

definition $Skip = (id{::}\ ('a \Rightarrow bool) \Rightarrow ('a \Rightarrow bool))$

lemma *assert-true-skip*: $\{.\top{::}'a \Rightarrow bool.\} = Skip$

lemma *skip-comp* [*simp*]: $Skip\ o\ S = S$

lemma *comp-skip*[*simp*]:$S\ o\ Skip = S$

lemma *assert-rel-skip*[*simp*]: $\{.\ \lambda\ (x,\ y)\ .\ True\ .\} = Skip$

lemma [*simp*]: $mono\ S \Longrightarrow mono\ S' \Longrightarrow mono\ (S\ o\ S')$

lemma [*simp*]: $mono\ \{.p{::}('a \Rightarrow bool).\}$

lemma [*simp*]: $mono\ [:r{::}('a \Rightarrow 'b \Rightarrow bool):]$

lemma *assert-true-skip-a*: $\{.\ x\ .\ True\ .\} = Skip$

lemma *assert-false-fail*: $\{.\bot{::}'a{::}boolean\text{-}algebra.\}\ = \bot$

lemma *magoc-comp*[*simp*]: $\top\ o\ S = \top$

lemma *left-comp*: $T\ o\ U = T'\ o\ U' \Longrightarrow S\ o\ T\ o\ U = S\ o\ T'\ o\ U'$

lemma *assert-demonic*: $\{.p.\}\ o\ [:r:] = \{.p.\}\ o\ [:x\ \rightsquigarrow\ y\ .\ p\ x \wedge r\ x\ y:]$

lemma $trs\ r \sqcap trs\ r' = trs\ (\lambda\ x\ y\ .\ inpt\ r\ x \wedge inpt\ r'\ x \wedge (r\ x\ y \vee r'\ x\ y))$

lemma *mono-assert*[*simp*]: $mono\ \{.p.\}$

lemma *mono-assume*[*simp*]: $mono\ [.p.]$

lemma *mono-demonic*[*simp*]: $mono\ [:r:]$

lemma *mono-comp-a*[*simp*]: $mono\ S \Longrightarrow mono\ T \Longrightarrow mono\ (S\ o\ T)$

lemma *mono-demonic-choice*[*simp*]: $mono\ S \Longrightarrow mono\ T \Longrightarrow mono\ (S \sqcap T)$

lemma *mono-Skip*[*simp*]: $mono\ Skip$

lemma *mono-comp*: $mono\ S \Longrightarrow S \le S' \Longrightarrow T \le T' \Longrightarrow S\ o\ T \le S'\ o\ T'$

lemma *sconjunctive-simp-a*: $sconjunctive\ S \Longrightarrow prec\ S = p \Longrightarrow rel\ S = r \Longrightarrow S = \{.p.\}\ o\ [:r:]$

lemma *sconjunctive-simp-b*: $sconjunctive\ S \Longrightarrow prec\ S = \top \Longrightarrow rel\ S = r \Longrightarrow S$

$= [:r:]$

lemma *sconj-Fail*[*simp*]: *sconjunctive Fail*

lemma *sconjunctive-simp-c*: *sconjunctive* $(S::('a \Rightarrow bool) \Rightarrow 'b \Rightarrow bool) \Longrightarrow prec$ $S = \bot \Longrightarrow S = Fail$

lemma *demonic-eq-skip*: $[: (=) :] = Skip$

definition $Havoc = [:\top:]$

definition $Magic = [:\bot::'a \Rightarrow 'b::boolean\text{-}algebra:]$

lemma *Magic-top*: $Magic = \top$

lemma [*simp*]: $Magic \neq Fail$

lemma *Havoc-Fail*[*simp*]: $Havoc\ o\ (Fail::'a \Rightarrow 'b \Rightarrow bool) = Fail$

lemma *demonic-havoc*: $[: \lambda x\ (x', y).\ True :] = Havoc$

lemma [*simp*]: *mono Magic*

lemma *demonic-false-magic*: $[: \lambda(x, y)\ (u, v).\ False :] = Magic$

lemma *demonic-magic*[*simp*]: $[:r:]\ o\ Magic = Magic$

lemma *magic-comp*[*simp*]: $Magic\ o\ S = Magic$

lemma *hvoc-magic*[*simp*]: $Havoc \circ Magic = Magic$

lemma $Havoc\ \top = \top$

lemma *Skip-id*[*simp*]: $Skip\ p = p$

lemma *demonic-pair-skip*: $[: x, y \rightsquigarrow u, v.\ x = u \wedge y = v :] = Skip$

lemma *comp-demonic-demonic*: $S\ o\ [:r:]\ o\ [:r':] = S\ o\ [:r\ OO\ r':]$

lemma *comp-demonic-assert*: $S\ o\ [:r:]\ o\ \{.p.\} = S\ o\ \{.\ x.\ \forall y\ .\ r\ x\ y \longrightarrow p\ y\ .\}$ $o\ [:r:]$

lemma *assert-demonic-eq-demonic*: $(\{.p.\}\ o\ [:r::'a \Rightarrow 'b \Rightarrow bool:] = [:r:]) = (\forall\ x$ $.\ p\ x)$

lemma *trs-inpt-top*: $inpt\ r = \top \Longrightarrow trs\ r = [:r:]$

8.3 Product and Fusion of predicate transformers

In this section we define the fusion and product operators from [?]. The fusion of two programs S and T is intuitively equivalent with the parallel execution of the two programs. If S and T assign nondeterministically some value to some program variable x, then the fusion of S and T will assign a value to x which can be assigned by both S and T.

definition *fusion* :: $(('a \Rightarrow bool) \Rightarrow ('b \Rightarrow bool)) \Rightarrow (('a \Rightarrow bool) \Rightarrow ('b \Rightarrow bool)) \Rightarrow (('a \Rightarrow bool) \Rightarrow ('b \Rightarrow bool))$ (**infixl** $\|$ *70*) **where**
$(S \parallel S')\ q\ x = (\exists\ (p::'a \Rightarrow bool)\ p'\ .\ p \sqcap p' \leq q \land S\ p\ x \land S'\ p'\ x)$

lemma *fusion-demonic*: $[:r:] \parallel [:r':] = [:r \sqcap r':]$

lemma *fusion-spec*: $(\{.p.\} \circ [:r:]) \parallel (\{.p'.\} \circ [:r':]) = (\{.p \sqcap p'.\} \circ [:r \sqcap r':])$

lemma *fusion-assoc*: $S \parallel (T \parallel U) = (S \parallel T) \parallel U$

lemma *fusion-refinement*: $S \leq T \Longrightarrow S' \leq T' \Longrightarrow S \parallel S' \leq T \parallel T'$

lemma *conjunctive* $S \Longrightarrow S \parallel \top = \top$

lemma *fusion-spec-local*: $a \in init \Longrightarrow ([:\ x \rightsquigarrow u,\ y\ .\ u \in init \land x = y\ :] \circ \{.p.\} \circ [:r:]) \parallel (\{.p'.\} \circ [:r':])$
$= [:\ x \rightsquigarrow u,\ y\ .\ u \in init \land x = y\ :] \circ \{.u,x\ .\ p\ (u,\ x) \land p'\ x.\} \circ [:u,\ x \rightsquigarrow y\ .\ r\ (u,\ x)\ y \land r'\ x\ y:]$ (**is** $?p \Longrightarrow ?S = ?T$)

lemma *fusion-demonic-idemp* [*simp*]: $[:r:] \parallel [:r:] = [:r:]$

lemma *fusion-spec-local-a*: $a \in init \Longrightarrow ([:x \rightsquigarrow u,\ y\ .\ u \in init \land x = y:] \circ \{.p.\} \circ [:r:]) \parallel [:r':]$
$= ([:x \rightsquigarrow u,\ y\ .\ u \in init \land x = y:] \circ \{.p.\} \circ [:u,\ x \rightsquigarrow y\ .\ r\ (u,\ x)\ y \land r'\ x\ y:])$

lemma *fusion-local-refinement*:
$a \in init \Longrightarrow (\bigwedge x\ u\ y\ .\ u \in init \Longrightarrow p'\ x \Longrightarrow r\ (u,\ x)\ y \Longrightarrow r'\ x\ y) \Longrightarrow$
$\{.p'.\}\ o\ (([:x \rightsquigarrow u,\ y\ .\ u \in init \land x = y:] \circ \{.p.\} \circ [:r:]) \parallel [:r':]) \leq [:x \rightsquigarrow u,\ y\ .\ u \in init \land x = y:] \circ \{.p.\} \circ [:r:]$

lemma *fusion-spec-demonic*: $(\{.p.\}\ o\ [:r:]) \parallel [:r':] = \{.p.\}\ o\ [:r \sqcap r':]$

definition *Fusion* :: $('c \Rightarrow (('a \Rightarrow bool) \Rightarrow ('b \Rightarrow bool))) \Rightarrow (('a \Rightarrow bool) \Rightarrow ('b \Rightarrow bool))$ **where**
$Fusion\ S\ q\ x = (\exists\ (p::'c \Rightarrow 'a \Rightarrow bool)\ .\ (INF\ c\ .\ p\ c) \leq q \land (\forall\ c\ .\ (S\ c)\ (p\ c)\ x))$

lemma *Fusion-spec*: $Fusion\ (\lambda\ n\ .\ \{.p\ n.\} \circ [:r\ n:]) = (\{.INFIMUM\ UNIV\ p.\} \circ [:INFIMUM\ UNIV\ r:])$

lemma *Fusion-demonic*: $Fusion\ (\lambda\ n\ .\ [:r\ n:]) = [:INF\ n\ .\ r\ n:]$

lemma *Fusion-refinement*: $(\bigwedge\ i\ .\ S\ i \leq T\ i) \Longrightarrow Fusion\ S \leq Fusion\ T$

lemma *mono-fusion*[*simp*]: $mono\ (S \parallel T)$

lemma *mono-Fusion*: $mono\ (Fusion\ S)$

definition $prod\text{-}pred\ A\ B = (\lambda(a,\ b).\ A\ a \wedge B\ b)$
definition $Prod :: (('a \Rightarrow bool) \Rightarrow ('b \Rightarrow bool)) \Rightarrow (('c \Rightarrow bool) \Rightarrow ('d \Rightarrow bool)) \Rightarrow (('a \times 'c \Rightarrow bool) \Rightarrow ('b \times 'd \Rightarrow bool))$
(**infixr** $**$ *70*)
where
$(S ** T)\ q = (\lambda\ (x,\ y)\ .\ \exists\ p\ p'\ .\ prod\text{-}pred\ p\ p' \leq q \wedge S\ p\ x \wedge T\ p'\ y)$

lemma *mono-prod*[*simp*]: $mono\ (S ** T)$

lemma *Prod-spec*: $(\{.p.\}\ o\ [:r:]) ** (\{.p'.\}\ o\ [:r':]) = \{.x,y\ .\ p\ x \wedge p'\ y.\}\ o\ [:x,\ y \rightsquigarrow u,\ v\ .\ r\ x\ u \wedge r'\ y\ v:]$

lemma *Prod-demonic*: $[:r:] ** [:r':] = [:x,\ y \rightsquigarrow u,\ v\ .\ r\ x\ u \wedge r'\ y\ v:]$

lemma *Prod-spec-Skip*: $(\{.p.\}\ o\ [:r:]) ** Skip = \{.x,y\ .\ p\ x.\}\ o\ [:x,\ y \rightsquigarrow u,\ v\ .\ r\ x\ u \wedge v = y:]$

lemma *Prod-Skip-spec*: $Skip ** (\{.p.\}\ o\ [:r:]) = \{.x,y\ .\ p\ y.\}\ o\ [:x,\ y \rightsquigarrow u,\ v\ .\ x = u \wedge r\ y\ v:]$

lemma *Prod-skip-demonic*: $Skip ** [:r:] = [:x,\ y \rightsquigarrow u,\ v\ .\ x = u \wedge r\ y\ v:]$

lemma *Prod-demonic-skip*: $[:r:] ** Skip = [:x,\ y \rightsquigarrow u,\ v\ .\ r\ x\ u \wedge\ y = v:]$

lemma *Prod-spec-demonic*: $(\{.p.\}\ o\ [:r:]) **\ [:r':] = \{.x,\ y\ .\ p\ x.\}\ o\ [:x,\ y \rightsquigarrow u,\ v\ .\ r\ x\ u \wedge r'\ y\ v:]$

lemma *Prod-demonic-spec*: $[:r:] ** (\{.p.\}\ o\ [:r':]) = \{.x,\ y\ .\ p\ y.\}\ o\ [:x,\ y \rightsquigarrow u,\ v\ .\ r\ x\ u \wedge r'\ y\ v:]$

lemma *pair-eq-demonic-skip*: $[:\ \lambda(x,\ y)\ (u,\ v).\ x = u \wedge v = y\ :] = Skip$

lemma *Prod-assert-skip*: $\{.p.\} ** Skip = \{.x,y\ .\ p\ x.\}$

lemma *Prod-skip-assert*: $Skip ** \{.p.\} = \{.x,y\ .\ p\ y.\}$

lemma *fusion-comute*: $S \parallel T = T \parallel S$

lemma *fusion-mono1*: $S \leq S' \Longrightarrow S \parallel T \leq S' \parallel T$

lemma *prod-mono1*: $S \le S' \Longrightarrow S ** T \le S' ** T$

lemma *prod-mono2*: $S \le S' \Longrightarrow T ** S \le T ** S'$

lemma *Prod-fusion*: $S ** T = ([:x,y \rightsquigarrow x' . x = x':] \; o \; S \; o \; [:x \rightsquigarrow x', y . x = x':]) \parallel ([:x, y \rightsquigarrow y' . y = y':] \; o \; T \; o \; [:y \rightsquigarrow x, y' . y = y':])$

lemma *refin-comp-right*: $(S::'a \Rightarrow 'b::order) \le T \Longrightarrow S \; o \; X \le T \; o \; X$

lemma *refin-comp-left*: $mono\ X \Longrightarrow (S::'a \Rightarrow 'b::order) \le T \Longrightarrow X \; o \; S \le X \; o \; T$

lemma *mono-angelic*[*simp*]: $mono\ \{:r:\}$

lemma [*simp*]: $Skip ** Magic = Magic$

lemma [*simp*]: $S ** Fail = Fail$

lemma [*simp*]: $Fail ** S = Fail$

lemma *demonic-conj*: $[:(r::'a \Rightarrow 'b \Rightarrow bool):] \; o \; (S \sqcap S') = ([:r:] \; o \; S) \sqcap ([:r:] \; o \; S')$

lemma *demonic-assume*: $[:r:] \; o \; [.p.] = [:x \rightsquigarrow y . r\ x\ y \wedge p\ y:]$

lemma *assume-demonic*: $[.p.] \; o \; [:r:] = [:x \rightsquigarrow y . p\ x \wedge r\ x\ y:]$

lemma [*simp*]: $(Fail::'a::boolean\text{-}algebra) \le S$

lemma *prod-skip-skip*[*simp*]: $Skip ** Skip = Skip$

lemma *fusion-prod*: $S \parallel T = [:x \rightsquigarrow y, z . x = y \wedge x = z:] \; o \; Prod\ S\ T \; o \; [:y , z \rightsquigarrow x . y = x \wedge z = x:]$

lemma [*simp*]: $prec\ S = \top \Longrightarrow prec\ T = \top \Longrightarrow prec\ (S ** T) = \top$

lemma *prec-skip*[*simp*]: $prec\ Skip = (\top::'a \Rightarrow bool)$

lemma [*simp*]: $prec\ S = \top \Longrightarrow prec\ T = \top \Longrightarrow prec\ (S \parallel T) = \top$

8.4 Functional Update

definition *update* :: $('a \Rightarrow 'b) \Rightarrow ('b \Rightarrow bool) \Rightarrow 'a \Rightarrow bool$ $([-\text{-}-])$ **where**
  $[-f-] = [:x \rightsquigarrow y . y = f\ x:]$
syntax
  -update :: *patterns* $\Rightarrow$ *tuple-args* $\Rightarrow$ *logic* $((1[- \text{ - } \rightsquigarrow \text{ - } -]))$
translations
  -update x (*-tuple-args* f F) == *CONST update* ((*-abs* x (*-tuple* f F)))

-update x (-tuple-arg F) == CONST update (-abs x F)

lemma *update-o-def*: $[-f\ o\ g-] = [-x \leadsto f\ (g\ x)-]$

lemma *update-simp*: $[-f-]\ q = (\lambda\ x\ .\ q\ (f\, x))$

lemma *update-assert-comp*: $[-f-]\ o\ \{.p.\} = \{.p\ o\ f.\}\ o\ [-f-]$

lemma *update-comp*: $[-f-]\ o\ [-g-] = [-g\ o\ f-]$

lemma *update-demonic-comp*: $[-f-]\ o\ [:r:] = [:x \leadsto y\ .\ r\ (f\, x)\ y:]$

lemma *demonic-update-comp*: $[:r:]\ o\ [-f-] = [:x \leadsto y\ .\ \exists\ z\ .\ r\ x\ z \wedge y = f\, z:]$

lemma *comp-update-demonic*: $S\ o\ [-f-]\ o\ [:r:] = S\ o\ [:x \leadsto y\ .\ r\ (f\, x)\ y:]$

lemma *comp-demonic-update*: $S\ o\ [:r:]\ o\ [-f-] = S\ o\ [:x \leadsto y\ .\ \exists\ z\ .\ r\ x\ z \wedge y = f\, z:]$

lemma *convert*: $(\lambda\ x\ y\ .\ (S::('a \Rightarrow bool) \Rightarrow ('b \Rightarrow bool))\ x\ (f\ y)) = [-f-]\ o\ S$

lemma *prod-update*: $[-f-] ** [-g-] = [-x,\ y \leadsto f\, x,\ g\ y\ -]$

lemma *prod-update-skip*: $[-f-] ** Skip = [-\ x,\ y \leadsto f\, x,\ y-]$

lemma *prod-skip-update*: $Skip ** [-f-] = [-\ x,\ y \leadsto x,\ f\, y-]$

lemma *prod-assert-update-skip*: $(\{.p.\}\ o\ [-f-]) ** Skip = \{.x{,}y\ .\ p\ x.\}\ o\ [-\ x,\ y \leadsto f\, x,\ y-]$

lemma *prod-skip-assert-update*: $Skip ** (\{.p.\}\ o\ [-f-]) = \{.x{,}y\ .\ p\ y.\}\ o\ [-\lambda\ (x,\ y)\ .\ (x,\ f\, y)-]$

lemma *prod-assert-update*: $(\{.p.\}\ o\ [-f-]) ** (\{.p'.\}\ o\ [-f'-]) = \{.x{,}y\ .\ p\ x \wedge p'\ y.\}\ o\ [-\lambda\ (x,\ y)\ .\ (f\, x,\ f'\ y)-]$

lemma *update-id-Skip*: $[-id-] = Skip$

lemma *prod-assert-assert-update*: $\{.p.\} ** (\{.p'.\}\ o\ [-f-]) = \{.x{,}y\ .\ p\ x \wedge p'\ y.\}\ o\ [-\ x,\ y \leadsto x,\ f\, y-]$

lemma *prod-assert-update-assert*: $(\{.p.\}\ o\ [-f-]) ** \{.p'.\} = \{.x{,}y\ .\ p\ x \wedge p'\ y.\}\ o\ [-\ x,\ y \leadsto f\, x,\ y-]$

lemma *prod-update-assert-update*: $[-f-] ** (\{.p.\}\ o\ [-f'-]) = \{.x{,}y\ .\ p\ y.\}\ o\ [-x,\ y \leadsto f\, x,\ f'\ y-]$

lemma *prod-assert-update-update*: $(\{.p.\}\ o\ [-f-]) ** [-f'-] = \{.x{,}y\ .\ p\ x\ .\}\ o\ [-$

$x,\ y \leadsto f\ x,\ f'\ y$−]

lemma *Fail-assert-update*: $Fail = \{.\bot.\}\ o\ [-\ (Eps\ \top)\ -]$

lemma *fail-assert-update*: $\bot = \{.\bot.\}\ o\ [-\ (Eps\ \top)\ -]$

lemma *update-fail*: $[-f-]\ o\ \bot = \bot$

lemma *fail-assert-demonic*: $\bot = \{.\bot.\}\ o\ [:\bot:]$

lemma *false-update-fail*: $\{.\lambda x.\ False.\}\ o\ [-f-] = \bot$

lemma *comp-update-update*: $S \circ [-f-] \circ [-f'-] = S \circ [-\ f'\ o\ f\ -]$

lemma *comp-update-assert*: $S \circ [-f-] \circ \{.p.\} = S \circ \{.p\ o\ f.\}\ o\ [-f-]$

lemma *prod-fail*: $\bot\ {*}{*}\ S = \bot$

lemma *fail-prod*: $S\ {*}{*}\ \bot = \bot$

lemma *assert-fail*: $\{.p{::}'a{::}boolean\text{-}algebra.\}\ o\ \bot = \bot$

lemma *angelic-assert*: $\{:r:\}\ o\ \{.p.\} = \{:x \leadsto y\ .\ r\ x\ y \wedge p\ y:\}$

lemma *Prod-Skip-angelic-demonic*: $Skip\ {*}{*}\ (\{:r:\}\ o\ [:r':]) = \{:s,x \leadsto s',y\ .\ r\ x\ y \wedge s' = s:\}\ o\ [:s,x \leadsto s',y\ .\ r'\ x\ y \wedge s' = s:]$

lemma *Prod-angelic-demonic-Skip*: $(\{:r:\}\ o\ [:r':])\ {*}{*}\ Skip = \{:x,\ u \leadsto y,\ u'\ .\ r\ x\ y \wedge u = u':\}\ o\ \ [:x,\ u \leadsto y,\ u'\ .\ r'\ x\ y \wedge u = u':]$

lemma *prec-rel-eq*: $p = p' \Longrightarrow r = r' \Longrightarrow \{.p.\}\ o\ [:r:] = \{.p'.\}\ o\ [:r':]$

lemma *prec-rel-le*: $p \le p' \Longrightarrow (\bigwedge\ x\ .\ p\ x \Longrightarrow r'\ x \le r\ x) \Longrightarrow \{.p.\}\ o\ [:r:] \le \{.p'.\}\ o\ [:r':]$

lemma *assert-update-eq*: $(\{.p.\}\ o\ [-f-] = \{.p'.\}\ o\ [-f'-]) = (p = p' \wedge (\forall\ x.\ p\ x \longrightarrow f\ x = f'\ x))$

lemma *update-eq*: $([-f-] = [-f'-]) = (f = f')$

lemma *spec-eq-iff*:
  shows *spec-eq-iff-1*: $p = p' \Longrightarrow f = f' \Longrightarrow \{.p.\}\ o\ [-f-] = \{.p'.\}\ o\ [-f'-]$
  and *spec-eq-iff-2*: $f = f' \Longrightarrow [-f-] = [-f'-]$
  and *spec-eq-iff-3*: $p = (\lambda\ x\ .\ True) \Longrightarrow f = f' \Longrightarrow \{.p.\}\ o\ [-f-] = [-f'-]$
  and *spec-eq-iff-4*: $p = (\lambda\ x\ .\ True) \Longrightarrow f = f' \Longrightarrow [-f-] = \{.p.\}\ o\ [-f'-]$

lemma *spec-eq-iff-a*:
  shows$(\bigwedge\ x\ .\ p\ x = p'\ x) \Longrightarrow (\bigwedge\ x\ .\ f\ x = f'\ x) \Longrightarrow \{.p.\}\ o\ [-f-] = \{.p'.\}\ o$

$[-f'-]$
  and $(\bigwedge x . f\,x = f'\,x) \Longrightarrow [-f-] = [-f'-]$
  and $(\bigwedge x . p\,x) \Longrightarrow (\bigwedge x . f\,x = f'\,x) \Longrightarrow \{.p.\}\ o\ [-f-] = [-f'-]$
  and $(\bigwedge x . p\,x) \Longrightarrow (\bigwedge x . f\,x = f'\,x) \Longrightarrow [-f-] = \{.p.\}\ o\ [-f'-]$

lemma *spec-eq-iff-prec*: $p = p' \Longrightarrow (\bigwedge x . p\,x \Longrightarrow f\,x = f'\,x) \Longrightarrow \{.p.\}\ o\ [-f-] = \{.p'.\}\ o\ [-f'-]$

lemma *trs-prod*: $trs\ r\ {**}\ trs\ r' = trs\ (\lambda\ (x,x')\ (y,y')\ .\ r\ x\ y \wedge r'\ x'\ y')$

lemma *sconjunctiveE*: $sconjunctive\ S \Longrightarrow (\exists\ p\ r\ .\ S = \{.\ p\ .\}\ o\ [:\ r :: 'a \Rightarrow 'b \Rightarrow bool:])$

lemma *sconjunctive-prod* [*simp*]: $sconjunctive\ S \Longrightarrow sconjunctive\ S' \Longrightarrow sconjunctive\ (S\ {**}\ S')$

lemma *nonmagic-prod* [*simp*]: $non\text{-}magic\ S \Longrightarrow non\text{-}magic\ S' \Longrightarrow non\text{-}magic\ (S\ {**}\ S')$

lemma *non-magic-comp* [*simp*]: $non\text{-}magic\ S \Longrightarrow non\text{-}magic\ S' \Longrightarrow non\text{-}magic\ (S\ o\ S')$

lemma *implementable-pred* [*simp*]: $implementable\ S \Longrightarrow implementable\ S' \Longrightarrow implementable\ (S\ {**}\ S')$

lemma *implementable-comp*[*simp*]: $implementable\ S \Longrightarrow implementable\ S' \Longrightarrow implementable\ (S\ o\ S')$

lemma *nonmagic-assert*: $non\text{-}magic\ \{.p::'a::boolean\text{-}algebra.\}$

8.5 Control Statements

definition $if\text{-}stm\ p\ S\ T = ([.p.]\ o\ S) \sqcap ([.-p.]\ o\ T)$

definition $while\text{-}stm\ p\ S = lfp\ (\lambda\ X\ .\ if\text{-}stm\ p\ (S\ o\ X)\ Skip)$

definition $Sup\text{-}less\ x\ (w::'b::wellorder) = Sup\ \{(x\ v)::'a::complete\text{-}lattice \mid v\ .\ v < w\}$

lemma *Sup-less-upper*: $v < w \Longrightarrow P\ v \le Sup\text{-}less\ P\ w$

lemma *Sup-less-least*: $(\bigwedge v\ .\ v < w \Longrightarrow P\ v \le Q) \Longrightarrow Sup\text{-}less\ P\ w \le Q$

theorem *fp-wf-induction*:
  $f\,x = x \Longrightarrow mono\ f \Longrightarrow (\forall\ w\ .\ (y\ w) \le f\ (Sup\text{-}less\ y\ w)) \Longrightarrow Sup\ (range\ y) \le x$

theorem *lfp-wf-induction*: $mono\ f \Longrightarrow (\forall\ w\ .\ (p\ w) \leq f\ (Sup\text{-}less\ p\ w)) \Longrightarrow Sup\ (range\ p) \leq lfp\ f$

theorem *lfp-wf-induction-a*: $mono\ f \Longrightarrow (\forall\ w\ .\ (p\ w) \leq f\ (Sup\text{-}less\ p\ w)) \Longrightarrow (SUP\ a.\ p\ a) \leq lfp\ f$

theorem *lfp-wf-induction-b*: $mono\ f \Longrightarrow (\forall\ w\ .\ (p\ w) \leq f\ (Sup\text{-}less\ p\ w)) \Longrightarrow S \leq (SUP\ a.\ p\ a) \Longrightarrow S \leq lfp\ f$

lemma [*simp*]: $mono\ S \Longrightarrow mono\ (\lambda X.\ if\text{-}stm\ b\ (S \circ X)\ T)$

definition $mono\text{-}mono\ F = (mono\ F \wedge (\forall\ f\ .\ mono\ f \longrightarrow mono\ (F\ f)))$

theorem *lfp-mono* [*simp*]:
$mono\text{-}mono\ F \Longrightarrow mono\ (lfp\ F)$

lemma *if-mono*[*simp*]: $mono\ S \Longrightarrow mono\ T \Longrightarrow mono\ (if\text{-}stm\ b\ S\ T)$

8.6 Hoare Total Correctness Rules

definition $Hoare\ p\ S\ q = (p \leq S\ q)$

definition $post\text{-}fun\ (p::'a::order)\ q = (if\ p \leq q\ then\ \top\ else\ \bot)$

lemma *post-mono* [*simp*]: $mono\ (post\text{-}fun\ p :: (\text{-}::\{order\text{-}bot,order\text{-}top\}))$

lemma *post-refin* [*simp*]: $mono\ S \Longrightarrow ((S\ p)::'a::bounded\text{-}lattice) \sqcap (post\text{-}fun\ p)\ x \leq S\ x$

lemma *post-top* [*simp*]: $post\text{-}fun\ p\ p = \top$

theorem *hoare-refinement-post*:
$mono\ f \Longrightarrow\ (Hoare\ x\ \ f\ y) = (\{.x::'a::boolean\text{-}algebra.\}\ o\ (post\text{-}fun\ y) \leq f)$

lemma *assert-Sup-range*: $\{.Sup\ (range\ (p::'W \Rightarrow 'a::complete\text{-}distrib\text{-}lattice)).\} = Sup(range\ (assert\ o\ p))$

lemma *Sup-range-comp*: $(Sup\ (range\ p))\ o\ S = Sup\ (range\ (\lambda\ w\ .\ ((p\ w)\ o\ S)))$

lemma *Sup-less-comp*: $(Sup\text{-}less\ P)\ w\ o\ S = Sup\text{-}less\ (\lambda\ w\ .\ ((P\ w)\ o\ S))\ w$

lemma *assert-Sup*: $\{.Sup\ (X::'a::complete\text{-}distrib\text{-}lattice\ set).\} = Sup\ (assert\ `\ X)$

lemma *Sup-less-assert*: $Sup\text{-}less\ (\lambda w.\ \{.\ (p\ w)::'a::complete\text{-}distrib\text{-}lattice\ .\})\ w = \{.Sup\text{-}less\ p\ w.\}$

lemma [*simp*]: *Sup-less* $(\lambda n\ x.\ t\ x = n)\ n = (\lambda\ x\ .\ (t\ x < n))$

lemma [*simp*]: *Sup-less* $(\lambda n.\ \{.x.\ t\ x = n.\} \circ S)\ n = \{.x.\ t\ x < n.\} \circ S$

lemma [*simp*]: $(SUP\ a.\ \{.x\ .t\ x = a.\} \circ S) = S$

theorem *hoare-fixpoint*:
mono-mono $F \Longrightarrow$
$(\forall\ f\ w\ .\ mono\ f \longrightarrow (Hoare\ (Sup\text{-}less\ p\ w)\ f\ y \longrightarrow Hoare\ ((p\ w)::'a \Rightarrow bool)\ (F\ f)\ y)) \Longrightarrow Hoare(Sup\ (range\ p))\ (lfp\ F)\ y$

theorem *hoare-sequential*:
$mono\ S \Longrightarrow (Hoare\ p\ (S\ o\ T)\ r) = (\ (\exists\ q.\ Hoare\ p\ S\ q \wedge Hoare\ q\ T\ r))$

theorem *hoare-choice*:
$Hoare\ \ p\ (S \sqcap T)\ q = (Hoare\ p\ S\ q \wedge Hoare\ p\ T\ q)$

theorem *hoare-assume*:
$(Hoare\ P\ [.R.]\ Q) = (P \sqcap R \le Q)$

lemma *hoare-if*: $mono\ S \Longrightarrow mono\ T \Longrightarrow Hoare\ (p \sqcap b)\ S\ q \Longrightarrow Hoare\ (p \sqcap -b)\ T\ q \Longrightarrow Hoare\ p\ (if\text{-}stm\ b\ S\ T)\ q$

lemma [*simp*]: $mono\ x \Longrightarrow mono\text{-}mono\ (\lambda X\ .\ if\text{-}stm\ b\ (x \circ X)\ Skip)$

lemma *hoare-while*:
$mono\ x \Longrightarrow (\forall\ \ w\ .\ Hoare\ ((p\ w) \sqcap b)\ x\ (Sup\text{-}less\ p\ w)) \Longrightarrow\ \ Hoare\ \ (Sup\ (range\ p))\ (while\text{-}stm\ b\ x)\ ((Sup\ (range\ p)) \sqcap -b)$

lemma *hoare-prec-post*: $mono\ S \Longrightarrow p \le p' \Longrightarrow q' \le q \Longrightarrow Hoare\ p'\ S\ q' \Longrightarrow Hoare\ p\ S\ q$

lemma [*simp*]: $mono\ x \Longrightarrow\ \ mono\ (while\text{-}stm\ b\ x)$

lemma *hoare-while-a*:
$mono\ x \Longrightarrow (\forall\ \ w\ .\ Hoare\ ((p\ w) \sqcap b)\ x\ (Sup\text{-}less\ p\ w)) \Longrightarrow p' \le\ \ (Sup\ (range\ p)) \Longrightarrow ((Sup\ (range\ p)) \sqcap -b) \le q$
$\Longrightarrow\ \ Hoare\ p'\ (while\text{-}stm\ b\ x)\ q$

lemma *hoare-update*: $p \le q\ o\ f \Longrightarrow Hoare\ p\ [-f-]\ q$

lemma *hoare-demonic*: $(\bigwedge\ x\ y\ .\ p\ x \Longrightarrow r\ x\ y \Longrightarrow q\ y) \Longrightarrow Hoare\ p\ [:r:]\ q$

lemma *refinement-hoare*: $S \le\ T \Longrightarrow Hoare\ (p::'a::order)\ S\ (q) \Longrightarrow Hoare\ p\ T\ q$

lemma *refinement-hoare-iff*: $(S \leq T) = (\forall\ p\ q\ .\ Hoare\ (p{::}'a{::}order)\ S\ (q) \longrightarrow Hoare\ p\ T\ q)$

8.7 Data Refinement

lemma *data-refinement*: $mono\ S' \Longrightarrow (\forall\ x\ a\ .\ \exists\ u\ .\ R\ x\ a\ u) \Longrightarrow$
$\{:x,\ a \leadsto x',\ u\ .\ x = x' \wedge R\ x\ a\ u:\}\ o\ S \leq S'\ o\ \{:y,\ b \leadsto y',\ v\ .\ y = y' \wedge R'\ y\ b\ v:\} \Longrightarrow$
$[:x \leadsto x',\ u\ .\ x = x':]\ o\ S\ o\ [:y,\ v \leadsto y'\ .\ y = y':]$
$\leq [:x \leadsto x',\ a\ .\ x = x':]\ o\ S'\ o\ [:y,\ b \leadsto y'\ .\ y = y':]$

lemma *mono-update*[*simp*]: $mono\ [-\ f\ -]$

end

9 Feedbackless HBD Translation

theory *FeedbacklessHBDTranslation* **imports** *Diagrams Refinement*
begin
context *BaseOperationFeedbacklessVars*
begin
definition *WhileFeedbackless* =
$while\text{-}stm\ (\lambda\ As\ .\ internal\ As \neq \{\})$
$[:As \leadsto As'\ .\ \exists\ A\ .\ A \in set\ As \wedge (out\ A) \in internal\ As \wedge\ As' = map\ (CompA\ A)\ (As \ominus [A]):]$

definition *TranslateHBDFeedbackless* = $WhileFeedbackless\ o\ [-(\lambda\ As\ .\ Parallel\text{-}list\ As)-]$

definition *ok-fbless* $As = (Deterministic\ As \wedge loop\text{-}free\ As \wedge Type\text{-}OK\ As)$

definition *TranslateHBDRec* = $\{.\ ok\text{-}fbless\ .\}$
$o\ [:As \leadsto As'\ .\ \exists\ L\ .\ perm\ (VarFB\ (Parallel\text{-}list\ As))\ L \wedge As' = fb\text{-}less\ L\ As:]$

lemma [*simp*]:$\{.As.\ length\ (VarFB\ (Parallel\text{-}list\ As)) = w.\}\ (TranslateHBDRec\ x)\ y \Longrightarrow [.\ -\ (\lambda As.\ internal\ As \neq \{\})\ .]\ x\ y$

lemma *internal-fb-less-step*: $loop\text{-}free\ As \Longrightarrow Type\text{-}OK\ As \Longrightarrow A \in set\ As \Longrightarrow out\ A \in internal\ As \Longrightarrow\ internal\ (fb\text{-}less\text{-}step\ A\ (As \ominus [A])) = internal\ As - \{out\ A\}$

lemma *ok-fbless-fb-less-step*: $ok\text{-}fbless\ As \Longrightarrow A \in set\ As \Longrightarrow out\ A \in internal\ As \Longrightarrow ok\text{-}fbless\ (fb\text{-}less\text{-}step\ A\ (As \ominus [A]))$

lemma *map-CompA-fb-out-less-step*: $Deterministic\ As \Longrightarrow$

loop-free As $\Longrightarrow$
Type-OK As $\Longrightarrow$ $A \in$ *set As* $\Longrightarrow$ *out A* $\in$ *internal As* $\Longrightarrow$ *map* (*CompA A*) (*As* $\ominus$ [*A*]) = *fb-out-less-step* (*out A*) *As*

lemma *length-diff*: $a \in$ *set x* $\Longrightarrow$ *length* ($x \ominus [a]$) < *length x*

thm *perm-cons*

lemma *perm-cons-a*: $\bigwedge$ *y* . $a \in$ *set x* $\Longrightarrow$ *distinct x* $\Longrightarrow$ *perm* ($x \ominus [a]$) *y* $\Longrightarrow$ *perm x* (*a* # *y*)

lemma [*simp*]: {.*As*. *length* (*VarFB* (*Parallel-list As*)) = *w*.} (*TranslateHBDRec x*) *y* $\Longrightarrow$
[. λ*As*. *internal As* $\neq$ {} .]
([:*As*$\rightsquigarrow$*As*$'$.$\exists$ *A*. $A \in$ *set As* $\wedge$ *out A* $\in$ *internal As* $\wedge$ *As*$'$ = *map* (*CompA A*) (*As* $\ominus$ [*A*]):]
({.*As*. *length* (*VarFB* (*Parallel-list As*)) < *w*.} (*TranslateHBDRec x*))) *y*

lemma *Feedbackless-Rec-While-refinement*: *TranslateHBDRec* $\leq$ *WhileFeedbackless*

lemma [*simp*]: *TranslateHBDRec o* [−(λ *As* . *Parallel-list As*)−] $\leq$ *TranslateHBDFeedbackless*

thm *FB-fb-less*(*1*)

lemma *Out-Parallel-fb-less*: $\bigwedge$ *As* . *Type-OK As* $\Longrightarrow$ *loop-free As* $\Longrightarrow$ *distinct L* $\Longrightarrow$ *set L* $\subseteq$ *internal As* $\Longrightarrow$
Out (*Parallel-list* (*fb-less L As*)) = *concat* (*map Out As*) $\ominus$ *L*

lemma *io-diagram-distinct-VarFB*: *io-diagram A* $\Longrightarrow$ *distinct* (*VarFB A*)

theorem *fbless-correctness*: *ok-fbless As* $\Longrightarrow$ *perm* (*VarFB* (*Parallel-list As*)) *L* $\Longrightarrow$
in-equiv (*FB* (*Parallel-list As*)) (*Parallel-list* (*fb-less L As*))

lemma *Hoare-TranslateHBDRec*: *Hoare* (λ *As* . *As* = *As-init* $\wedge$ *ok-fbless As*)
(*TranslateHBDRec o* [−(λ *As* . *Parallel-list As*)−])
(λ *A* . *in-equiv* (*FB* (*Parallel-list As-init*)) *A*)

theorem *TranslateHBDFeedbacklessCorrectness*: *Hoare* (λ *As* . *As* = *As-init* $\wedge$ *ok-fbless As*)
TranslateHBDFeedbackless
(λ *A* . *in-equiv* (*FB* (*Parallel-list As-init*)) *A*)

end

end

10 Properties for Proving the Abstract Translation Algorithm

theory *HBDTranslationProperties* **imports** *ExtendedHBDAlgebra Diagrams*
begin
context *BaseOperationVars*
begin

lemma *io-diagram-fb-perm-eq*: *io-diagram A* $\Longrightarrow$ *fb-perm-eq A*

theorem *FeedbackSerial*: *io-diagram A* $\Longrightarrow$ *io-diagram B* $\Longrightarrow$ *set* (*In A*) $\cap$ *set* (*In B*) = {} (**required**)
$\Longrightarrow$ *set* (*Out A*) $\cap$ *set* (*Out B*) = {} $\Longrightarrow$ *FB* (*A* ||| *B*) = *FB* (*FB* (*A*) ;; *FB* (*B*))

lemmas *fb-perm-sym* = *fb-perm* [*THEN sym*]

declare *length-TVs* [*simp del*]

declare [[*simp-trace-depth-limit=40*]]

lemma *in-out-equiv-FB*: *io-diagram B* $\Longrightarrow$ *in-out-equiv A B* $\Longrightarrow$ *in-out-equiv* (*FB A*) (*FB B*)

end

end

11 HBD Translation Algorithms that use Feedback Composition

theory *HBDTranslationsUsingFeedback* **imports** *HBDTranslationProperties Refinement*
begin

context *BaseOperationVars*
begin

definition *TranslateHBD* =
while-stm (λ *As* . *length As* > *1*)(
[:*As* $\rightsquigarrow$ *As'* . $\exists$ *Bs Cs* . *1* < *length Bs* $\wedge$ *perm As* (*Bs* @ *Cs*) $\wedge$ *As'* = *FB* (*Parallel-list Bs*) # *Cs*:]
$\sqcap$

$[:As \leadsto As'$. $\exists$ A B Bs . *perm* As $(A \# B \# Bs) \land As' = (FB\ (FB\ A\ ;;\ FB\ B)) \# Bs:]$
)
o $[-(\lambda\ As\ .\ FB(As\ !\ 0))-]$

lemma [*simp*]:*Suc* $0 \leq$ *length As-init* $\Longrightarrow$
Hoare (λAs. *in-out-equiv* (*FB* (*As* ! *0*)) (*FB* (*Parallel-list As-init*))) [$-\lambda As$. *FB* (*As* ! *0*)$-$] (λS. *in-out-equiv S* (*FB* (*Parallel-list As-init*)))

definition *invariant As-init n As* = (*length As* = *n* $\land$ *io-distinct As* $\land$ *in-out-equiv* (*FB* (*Parallel-list As*)) (*FB* (*Parallel-list As-init*)) $\land$ $n \geq 1$)

lemma *io-diagram-Parallel-list*: $\forall$ $A \in$ *set As* . *io-diagram A* $\Longrightarrow$ *distinct* (*concat* (*map Out As*)) $\Longrightarrow$ *io-diagram* (*Parallel-list As*)

lemma *io-diagram-Parallel-list-a*: *io-distinct As* $\Longrightarrow$ *io-diagram* (*Parallel-list As*)

thm *Parallel-list-cons*

thm *Parallel-assoc-gen*

thm *ParallelId-left*
thm *io-diagram-Parallel-list*

lemma *Parallel-list-append*: $\forall$ $A \in$ *set As* . *io-diagram A* $\Longrightarrow$ *distinct* (*concat* (*map Out As*)) $\Longrightarrow$ $\forall$ $A \in$ *set Bs* . *io-diagram A*
$\Longrightarrow$ *distinct* (*concat* (*map Out Bs*))$\Longrightarrow$
Parallel-list (*As* @ *Bs*) = *Parallel-list As* ||| *Parallel-list Bs*

primrec *sequence* :: *nat* $\Rightarrow$ *nat list* **where**
sequence 0 = [] |
sequence (*Suc n*) = *sequence n* @ [*n*]

lemma *sequence* (*Suc* (*Suc 0*)) = [*0*,*1*]

lemma *in-out-equiv-io-diagram*[*simp*]: *in-out-equiv A B* $\Longrightarrow$ *io-diagram B* $\Longrightarrow$ *io-diagram A*

thm *comp-parallel-distrib*

lemma *in-out-equiv-Parallel-cong-right*: *io-diagram A* $\Longrightarrow$ *io-diagram C* $\Longrightarrow$ *set* (*Out A*) $\cap$ *set* (*Out B*) = {} $\Longrightarrow$ *in-out-equiv B C*
$\Longrightarrow$ *in-out-equiv* (*A* ||| *B*) (*A* ||| *C*)

lemma *perm-map*: *perm x y* $\Longrightarrow$ *perm* (*map f x*) (*map f y*)

lemma *distinct-concat-perm*: $\bigwedge$ Y . *distinct* (*concat* X) $\Longrightarrow$ *perm* X Y $\Longrightarrow$ *distinct* (*concat* Y)

lemma *distinct-Par-equiv-a*: $\bigwedge$ *Bs* . $\forall$ $A \in$ *set As* . *io-diagram* $A \Longrightarrow$ *distinct* (*concat* (*map Out As*)) $\Longrightarrow$ *perm As Bs* $\Longrightarrow$
in-out-equiv (*Parallel-list As*) (*Parallel-list Bs*)

thm *distinct-concat-perm*
thm *perm-map*

lemma *distinct-FB*: *distinct* (*In A*) $\Longrightarrow$ *distinct* (*In* (*FB A*))

lemma *io-distinct-FB-cat*: *io-distinct* (*A # Cs*) $\Longrightarrow$ *io-distinct* (*FB A # Cs*)

lemma *io-distinct-perm*: *io-distinct As* $\Longrightarrow$ *perm As Bs* $\Longrightarrow$ *io-distinct Bs*

lemma [*simp*]: *distinct* (*concat* X) $\Longrightarrow$ *op-list* [] ($\oplus$) (X) = *concat* X

lemma [*simp*]: *io-distinct As* $\Longrightarrow$ *perm As* (*Bs* @ *Cs*) $\Longrightarrow$ *io-distinct* (*FB* (*Parallel-list Bs*) # *Cs*)

lemma *io-distinct-append-a*: *io-distinct As* $\Longrightarrow$ *perm As* (*Bs* @ *Cs*) $\Longrightarrow$ *io-distinct Bs*

lemma *io-distinct-append-b*: *io-distinct As* $\Longrightarrow$ *perm As* (*Bs* @ *Cs*) $\Longrightarrow$ *io-distinct Cs*

lemma [*simp*]: *io-distinct As* $\Longrightarrow$ *perm As* (*Bs* @ *Cs*) $\Longrightarrow$ *io-diagram* (*FB* (*FB* (*Parallel-list Bs*) ||| *Parallel-list Cs*))

lemma [*simp*]: *io-distinct As* $\Longrightarrow$ *io-diagram* (*FB* (*Parallel-list As*))

lemma *io-distinct-set-In*[*simp*]: *io-distinct x* $\Longrightarrow$ *perm x* (*A # B # Bs*) $\Longrightarrow$ *set* (*In A*) $\cap$ *set* (*In B*) = {}

lemma *io-distinct-set-Out*[*simp*]: *io-distinct x* $\Longrightarrow$ *perm x* (*A # B # Bs*) $\Longrightarrow$ *set* (*Out A*) $\cap$ *set* (*Out B*) = {}

lemma *distinct-Par-equiv-b*: *io-distinct As* $\Longrightarrow$ *perm As* (*Bs* @ *Cs*) $\Longrightarrow$ *in-out-equiv* (*FB* (*FB* (*Parallel-list Bs*) ||| *Parallel-list Cs*)) (*FB* (*Parallel-list As*))

lemma *distinct-Par-equiv*: *io-distinct As-init* $\Longrightarrow$ *Suc 0* $\leq$ *length As-init* $\Longrightarrow$
length As = w $\Longrightarrow$ *io-distinct As* $\Longrightarrow$ *in-out-equiv* (*FB* (*Parallel-list As*)) (*FB* (*Parallel-list As-init*)) $\Longrightarrow$
Suc 0 < w $\Longrightarrow$ *Suc 0* < *length Bs* $\Longrightarrow$ *perm As* (*Bs* @ *Cs*) $\Longrightarrow$
io-distinct (*FB* (*Parallel-list Bs*) # *Cs*) $\wedge$ *in-out-equiv* (*FB* (*FB* (*Parallel-list Bs*) ||| *Parallel-list Cs*)) (*FB* (*Parallel-list As-init*))

lemma *AAAA-x*[*simp*]: *io-distinct As-init* $\Longrightarrow$ *Suc 0* $\leq$ *length As-init* $\Longrightarrow$ *invariant As-init w x* $\Longrightarrow$ *Suc 0* $<$ *length x* $\Longrightarrow$ *Suc 0* $<$ *length Bs*
$\Longrightarrow$ *perm x* (*Bs* @ *Cs*)
$\Longrightarrow$ *invariant As-init* (*Suc* (*length Cs*)) (*FB* (*Parallel-list Bs*) # *Cs*)

term {*1*,*2*,*3*} − {*2*,*3*}

thm *ParallelId-right*

lemma [*simp*]: *io-distinct As-init* $\Longrightarrow$
Suc 0 $\leq$ *length As-init* $\Longrightarrow$ *invariant As-init w x* $\Longrightarrow$ *Suc 0* $<$ *length x* $\Longrightarrow$ *perm x* (*A* # *B* # *Bs*)
$\Longrightarrow$ *invariant As-init* (*Suc* (*length Bs*)) (*FB* (*FB A* ;; *FB B*) # *Bs*)

lemma [*simp*]: *io-distinct As-init* $\Longrightarrow$ *Suc 0* $\leq$ *length As-init* $\Longrightarrow$
Hoare (*invariant As-init w* $\sqcap$ (λ*As*. *Suc 0* $<$ *length As*))
[:*As*$\rightsquigarrow$*As'*.$\exists$ *Bs*. *Suc 0* $<$ *length Bs* $\wedge$ ($\exists$ *Cs*. *perm As* (*Bs* @ *Cs*) $\wedge$ *As'* = *FB* (*Parallel-list Bs*) # *Cs*):] (*Sup-less* (*invariant As-init*) *w*)

lemma [*simp*]: *io-distinct As-init* $\Longrightarrow$ *Suc 0* $\leq$ *length As-init* $\Longrightarrow$
Hoare (*invariant As-init w* $\sqcap$ (λ*As*. *Suc 0* $<$ *length As*))
[:*As*$\rightsquigarrow$*As'*.$\exists$ *A B Bs*. *perm As* (*A* # *B* # *Bs*) $\wedge$ *As'* = *FB* (*FB A* ;; *FB B*) # *Bs*:] (*Sup-less* (*invariant As-init*) *w*)

theorem *CorrectnessTranslateHBD*: *io-distinct As-init* $\Longrightarrow$ *length As-init* $\geq$ *1* $\Longrightarrow$
Hoare (*io-distinct* $\sqcap$ (λ *As* . *As* = *As-init*)) *TranslateHBD* (λ *S* . *in-out-equiv S* (*FB* (*Parallel-list As-init*)))
end

end