Mechanically Proving Determinacy of Hierarchical Block Diagram Translations

Viorel Preoteasa, Aalto University, Finland and Iulia Dragomir, Verimag, France

July 8, 2017

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**
$[] \otimes y = []\ |$
$(a\ \#\ x) \otimes y = (\mathit{if}\ a \in set\ y\ \mathit{then}\ a\ \#\ (x \otimes y)\ \mathit{else}\ x \otimes y)$

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*: $\mathit{set}\ x \subseteq \mathit{set}\ y \Longrightarrow x \ominus y = []$

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

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

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

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

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*: $\mathit{set}\ x \cap \mathit{set}\ u = \{\} \Longrightarrow x \ominus y \ominus (z \ominus u) = (x \ominus y \ominus z)$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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*: $set\ x = \{\} \Longrightarrow x = []$

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

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

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

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

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

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*: $perm\ x\ x' \Longrightarrow perm\ y\ y' \Longrightarrow f = op \otimes \vee f = op \ominus \vee f = op \oplus \Longrightarrow perm\ (f\ x\ y)\ (f\ x'\ y')$

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

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

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

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

theory *AlgebraFeedbackless* **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 $TI\text{-}comp[simp]$: $TI\ S' = TO\ S \Longrightarrow TI\ (S\ oo\ S') = TI\ S$
assumes $TO\text{-}comp[simp]$: $TI\ S' = TO\ S \Longrightarrow TO\ (S\ oo\ S') = TO\ S'$
assumes $comp\text{-}id\text{-}left\ [simp]$: $ID\ (TI\ S)\ oo\ S = S$
assumes $comp\text{-}id\text{-}right\ [simp]$: $S\ oo\ ID\ (TO\ S) = S$
assumes $comp\text{-}assoc$: $TI\ T = TO\ S \Longrightarrow TI\ R = TO\ T \Longrightarrow S\ oo\ T\ oo\ R = S\ oo\ (T\ oo\ R)$

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

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

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

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

assumes $TI\text{-}Sink[simp]$: $TI\ (Sink\ ts) = ts$
assumes $TO\text{-}Sink[simp]$: $TO\ (Sink\ ts) = []$

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

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

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

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 $fbtype\ S\ tsa\ ts\ ts' = (TI\ S = tsa @ ts \wedge TO\ S = tsa @ ts')$

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* ^^(*length tsa*)) (*S* $\parallel$ *T*) = ((*fb* ^^(*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* ^^(*length tsa*)) ((*ID tsa* ∥ *A*) *oo S oo* (*ID tsa* ∥ *B*)) = *A oo* (*fb* ^^(*length tsa*)) *S oo B*

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

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

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

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

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

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

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

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

set-var $(a \# x)$ b = (*if* $a = b$ *then ID* [*TV a*] ∥ *Sink* (*TVs x*) *else Sink* [*TV a*] ∥ *set-var x b*)

lemma *TO-set-var*[*simp*]: *TO* (*set-var x a*) = [*TV a*]

lemma *TI-set-var*[*simp*]: *TI* (*set-var x a*) = *TVs x*

primrec *switch* :: *'var list* ⇒ *'var list* ⇒ *'a* ([- ⇝ -]) **where**
[*x* ⇝ []] = *Sink* (*TVs x*) |
[*x* ⇝ *a* # *y*] = *Split* (*TVs x*) *oo set-var x a* ∥ [*x* ⇝ *y*]

lemma *TI-switch*[*simp*]: *TI* [*x* ⇝ *y*] = *TVs x*

lemma *TO-switch*[*simp*]: *TO* [*x* ⇝ *y*] = *TVs y*

lemma *switch-not-in-Sink*: *a* ∉ *set y* ⟹ [*a* # *x* ⇝ *y*] = *Sink* [*TV a*] ∥ [*x* ⇝ *y*]

lemma *distinct-id*: *distinct x* ⟹ [*x* ⇝ *x*] = *ID* (*TVs x*)

lemma *set-var-nin*: *a* ∉ *set x* ⟹ *set-var* (*x* @ *y*) *a* = *Sink* (*TVs x*) ∥ *set-var y a*

lemma *set-var-in*: *a* ∈ *set x* ⟹ *set-var* (*x* @ *y*) *a* = *set-var x a* ∥ *Sink* (*TVs y*)

lemma *set-var-not-in*: *a* ∉ *set y* ⟹ *set-var y a* = *Arb* (*TV a*) ∥ *Sink* (*TVs y*)

lemma *set-var-in-a*: *a* ∉ *set y* ⟹ *set-var* (*x* @ *y*) *a* = *set-var x a* ∥ *Sink* (*TVs y*)

lemma *switch-append*: [*x* ⇝ *y* @ *z*] = *Split* (*TVs x*) *oo* [*x* ⇝ *y*] ∥ [*x* ⇝ *z*]

lemma *switch-nin-a-new*: *set x* ∩ *set y'* = {} ⟹ [*x* @ *y* ⇝ *y'*] = *Sink* (*TVs x*) ∥ [*y* ⇝ *y'*]

lemma *switch-nin-b-new*: *set y* ∩ *set z* = {} ⟹ [*x* @ *y* ⇝ *z*] = [*x* ⇝ *z*] ∥ *Sink* (*TVs y*)

lemma *var-switch*: *distinct* (*x* @ *y*) ⟹ [*x* @ *y* ⇝ *y* @ *x*] = *Switch* (*TVs x*) (*TVs y*)

lemma *switch-par*: *distinct* (*x* @ *y*) ⟹ *distinct* (*u* @ *v*) ⟹ *TI S* = *TVs x* ⟹ *TI T* = *TVs y* ⟹ *TO S* = *TVs v* ⟹ *TO T* = *TVs u* ⟹
S ∥ *T* = [*x* @ *y* ⇝ *y* @ *x*] *oo T* ∥ *S oo* [*u* @ *v* ⇝ *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*: $distinct\ x \Longrightarrow perm\ x\ y \Longrightarrow set\ z \subseteq set\ y \Longrightarrow [x \rightsquigarrow y]\ oo\ [y \rightsquigarrow z] = [x \rightsquigarrow z]$

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

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* ∥ [[] $\rightsquigarrow$ []] = *A*

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

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

lemma *comp-id-switch*[*simp*]: *distinct x* $\Longrightarrow$ *TI S* = *TVs x* $\Longrightarrow$ [*x* $\rightsquigarrow$ *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*] ∥ (*A oo B*) = ([*x* $\rightsquigarrow$ *x*] ∥ *A*) *oo* ([*x* $\rightsquigarrow$ *x*] ∥ *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*: $\bigwedge A\ B\ v\ x\ .\ distinct\ (u\ @\ v\ @\ x) \Longrightarrow TO\ A = TVs\ (v@x) \Longrightarrow TI\ B = TVs\ (u\ @\ x) \Longrightarrow\ TVs\ u = TVs\ v$
$\Longrightarrow (fb\ \hat{}\,\hat{}\ length\ u)\ (([u \rightsquigarrow u] \parallel A)\ oo\ [u\ @\ v\ @\ x \rightsquigarrow v\ @\ u\ @\ x]\ oo\ ([v \rightsquigarrow v] \parallel B)) = A\ oo\ B$

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

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* $(x @ y) \wedge$ *distinct* $(u @ v) \wedge$ *TVs* $x =$ *TI* A
$\wedge$ *TVs* $u =$ *TO* $A \wedge$ *TVs* $y =$ *TI* $B \wedge$
TVs $v =$ *TO* $B \wedge A \parallel B = [x @ y \rightsquigarrow y @ x]$ *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* $b \Longrightarrow [a \rightsquigarrow a @ a] = [b \rightsquigarrow b @ b]$

2.1 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 $S = Split\ (TI\ S)\ oo\ (S\ oo\ [\ a\ \#\ x \leadsto [a]\ ]) \parallel (S\ oo\ [\ a\ \#\ x \leadsto x\ ])$

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

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

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

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

3 Abstract Algebra of Hierarchical Block Diagrams with all axioms

theory *AlgebraFeedback* **imports** *AlgebraFeedbackless*
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 ^^ (2::nat)) ([a # b # x $\rightsquigarrow$ b # a # x] oo S oo [b # a # y $\rightsquigarrow$ a # b # y]) = (fb ^^ (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 ^^ (Suc (length z))) ([a # z @ x $\rightsquigarrow$ z @ a # x] oo S oo [z @ a # y $\rightsquigarrow$ a # z @ y]) = (fb ^^ (Suc (length z))) S*

lemma *swap-power*: *(f ^^ n) ((f ^^ m) S) = (f ^^ m) ((f ^^ 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 ^^ (length (u @ v))) ([u @ v @ x $\rightsquigarrow$ v @ u @ x] oo S oo [v @ u @ y $\rightsquigarrow$ u @ v @ y]) = (fb ^^ (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 ^^ (length u)) ([v @ x $\rightsquigarrow$ u @ x] oo S oo [u @ y $\rightsquigarrow$ v @ y]) = (fb ^^ (length u)) S*

end

end

4 Diagrams with Named Inputs and Outputs

theory *DiagramFeedbackless* **imports** *AlgebraFeedbackless*
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 *Var A B = (Out A)* $\otimes$ *(In B)*

definition *io-diagram A = (TVs (In A) = TI (Trs A)* $\wedge$ *TVs (Out A) = TO (Trs A)* $\wedge$ *distinct (In A)* $\wedge$ *distinct (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*: $io\text{-}diagram\ A \Longrightarrow io\text{-}diagram\ B\ \Longrightarrow set\ (Out\ A) \cap set\ (Out\ B) = \{\} \Longrightarrow io\text{-}diagram\ (A\ |||\ B)$

lemma *Parallel-indep*: $io\text{-}diagram\ A \Longrightarrow io\text{-}diagram\ B\ \Longrightarrow set\ (In\ A) \cap set\ (In\ B) = \{\} \Longrightarrow$

$A \mathbin{|||} B = (\!| In = In\ A\ @\ In\ B,\ Out = Out\ A\ @\ Out\ B,\ Trs = (Trs\ A \parallel Trs\ B)\ |\!)$

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

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**
$FB\ A = (let\ I = In\ A \ominus Var\ A\ A\ in\ let\ O' = Out\ A \ominus Var\ A\ A\ in$
$(\!| In = I,\ Out = O',\ Trs = (fb\ \hat{}\ \hat{}\ (length\ (Var\ A\ A)))\ ([Var\ A\ A\ @\ I \rightsquigarrow In\ A]\ oo\ Trs\ A\ oo\ [Out\ A \rightsquigarrow Var\ A\ A\ @\ O'])\ |\!))$

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

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

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

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

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*: $io\text{-}diagram\ A \Longrightarrow FB\ (FB\ A) = FB\ A$

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 $in\text{-}equiv\ A\ B = (perm\ (In\ A)\ (In\ B) \wedge Trs\ A = [In\ A \rightsquigarrow In\ B]$

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 $\square$ = (|*In* = [], *Out* = [], *Trs* = *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* []) (*op* $\|$)

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

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-OK* *As* = (($\forall$ $B \in$ *set* *As* . *io-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-OK* *As* = (($\forall$ $B \in$ *set* *As* . *io-diagram* B $\wedge$ *length* (*Out* B) = *1*) $\wedge$ *distinct* (*map* *out* *As*))

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

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

primrec *get-other-out* :: $'c \Rightarrow ('c,\ 'd)\ Dgr\ list \Rightarrow ('c,\ 'd)\ Dgr\ list$ **where**
get-other-out $x\ [] = []$ |
get-other-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-less-step A As* = *map* (*CompA A*) *As*

definition *fb-out-less-step x As* = *fb-less-step* (*get-comp-out x As*) (*get-other-out x As*)

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

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

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

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

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

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

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

lemma [*simp*]: *Parallel-list* (*A* # *As*) = (*A* ||| *Parallel-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-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-OK As* $\Longrightarrow$ $A \in$ *set As* $\Longrightarrow$ *Out A* = [*out A*]

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

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

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

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

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

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

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

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

lemma *concat-map-Out-get-other-out*: *Type-OK As* $\Longrightarrow$ *concat* (*map Out* (*get-other-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*: $\mathit{Type\text{-}OK}\ (A \# As) \Longrightarrow A \notin \mathit{set}\ As$

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

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

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

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

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

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

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

thm *union-set-In-CompA*

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

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

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

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

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

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

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

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

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

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

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

lemma *parallel-list-cons*: $parallel\text{-}list\ (A\ \#\ As) = A \parallel parallel\text{-}list\ As$

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

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

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

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

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

lemma *io-diagram-ParallelId*[*simp*]: *io-diagram* □

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

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*]: $length\ (Out\ A) = 1 \Longrightarrow out\ A \in set\ (In\ B) \Longrightarrow CompA\ A\ B = A\ ;;\ B$

lemma *CompA-not-in*[*simp*]: $length\ (Out\ A) = 1 \Longrightarrow out\ A \notin set\ (In\ B) \Longrightarrow CompA\ A\ B = B$

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

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' ∩ *set* z = {} ⟹ *TVs* x = *TVs* y ⟹ *TVs* x' = *TVs* y' ⟹ *Subst* (x @ x') (y @ y') z = *Subst* x y z

lemma [*simp*]: *set* x ∩ *set* z = {} ⟹ *TVs* x = *TVs* y ⟹ *TVs* x' = *TVs* y' ⟹ *Subst* (x @ x') (y @ y') z = *Subst* x' y' z

lemma [*simp*]: *set* x ∩ *set* z = {} ⟹ *TVs* x = *TVs* y ⟹ *Subst* x y z = z

lemma [*simp*]: *distinct* x ⟹ *TVs* x = *TVs* y ⟹ *Subst* x y x = y

lemma *TVs* x = *TVs* y ⟹ *length* x = *length* y

thm *length-TVs*

lemma *in-equiv-switch-Parallel*: *io-diagram* A ⟹ *io-diagram* B ⟹ *set* (*Out* A) ∩ *set* (*Out* B) = {} ⟹
in-equiv (A ||| B) ((B ||| A) ;; [[*Out* B @ *Out* A ⇝ *Out* A @ *Out* B]])

lemma *in-out-equiv-Parallel*: *io-diagram* A ⟹ *io-diagram* B ⟹ *set* (*Out* A) ∩ *set* (*Out* B) = {} ⟹ *in-out-equiv* (A ||| B) (B ||| A)

declare *Subst-eq* [*simp*]

lemma assumes *in-equiv* A A' **shows** [*simp*]: *perm* (*In* A) (*In* A')

lemma *Subst-cancel-left-type*: *set* x ∩ *set* z = {} ⟹ *TVs* x = *TVs* y ⟹ *Subst* (x @ z) (y @ z) w = *Subst* x y w

lemma *diff-eq-set-right*: *set* y = *set* z ⟹ (x ⊖ y) = (x ⊖ z)

lemma [*simp*]: *set* (y ⊖ x) ∩ *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*: $deterministic\ (Trs\ A) \Longrightarrow length\ (Out\ A) = 1 \Longrightarrow io\text{-}diagram\ A \Longrightarrow io\text{-}diagram\ B \Longrightarrow io\text{-}diagram\ C \Longrightarrow$
$(*set\ (Out\ B) \cap set\ (Out\ C) = \{\} \Longrightarrow (*from\ in\text{-}equiv\text{-}CompA\text{-}Parallel\text{-}b\ *)*)$
$in\text{-}equiv\ (CompA\ A\ B\ |||\ CompA\ A\ C)\ (CompA\ A\ (B\ |||\ C))$

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*: $distinct\ x \Longrightarrow distinct\ u \Longrightarrow set\ y \subseteq set\ x \Longrightarrow set\ v \subseteq set\ u \Longrightarrow TVs\ x = TVs\ u$
$\Longrightarrow Subst\ x\ u\ y = v \Longrightarrow [x \leadsto y] = [u \leadsto v]$

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

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

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

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*: $Type\text{-}OK\ As \Longrightarrow distinct\ As$

lemma *TI-parallel-list-a*: $TI\ (parallel\text{-}list\ As) = concat\ (map\ TI\ As)$

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

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*) $\Longrightarrow$ $C =$ *CompA A* (*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*: *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*: $\bigwedge$ *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*)

thm *fb-CompA*

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

end

5 Porperties used for proving the Feedbackless algorithm

theory *Feedbackless* **imports** *DiagramFeedbackless*
begin
  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*)))$\}$

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

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

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

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

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

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

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

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

thm *in-equiv-fb-fb-less-step*
thm *fb-CompA*
thm *fb-less-step-compA*
thm *fb-out-less-step-def*

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

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

end

end

6 The order of internal states does not change the result of feedbackless

theory *FeedbacklessPerm* **imports** *Feedbackless*
begin
context *BaseOperationFeedbacklessVars*
begin

lemma [*simp*]: *out B* $\notin$ *set* (*In A*) $\Longrightarrow$ *CompA B A* = *A*

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

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'*] $\|$ [*z* $\rightsquigarrow$ *z'*] $\|$ [*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 $[x \rightsquigarrow y\ @\ z\ @\ t]\ oo\ [y' \rightsquigarrow u] \parallel [z' \rightsquigarrow v] \parallel [t' \rightsquigarrow w] = [x \rightsquigarrow Subst\ y'\ y\ u\ @\ Subst\ z'\ z\ v\ @\ Subst\ t'\ t\ w]$

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

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

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

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

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*]: $io\text{-}diagram\ A$
and [*simp*]: $io\text{-}diagram\ B$
and [*simp*]: $io\text{-}diagram\ C$
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 $in\text{-}equiv\ (CompA\ (CompA\ B\ A)\ (CompA\ B\ C))\ (CompA\ (CompA\ A\ B)\ (CompA\ A\ C))$

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

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

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

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

thm *in-equiv-tran*

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

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

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

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

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

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

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

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

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

lemma *In-Equiv-split*: $\bigwedge$ *Bs* . *In-Equiv As Bs* $\Longrightarrow$ *A* $\in$ *set As*
$\Longrightarrow$ $\exists$ *B As' As'' Bs' Bs''* . *As* = *As'* @ *A* # *As''* $\wedge$ *Bs* = *Bs'* @ *B* # *Bs''* $\wedge$ *in-equiv A B* $\wedge$ *In-Equiv As' Bs'* $\wedge$ *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* $\in$ *internal As*
shows *In-Equiv* (*fb-out-less-step a As*) (*fb-out-less-step a Bs*)

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

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

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

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

lemma *Type-OK-fb-less*: $\bigwedge$ *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*)
end

end

7 Properties for proving the nondeterministic algorithm

theory *Feedback* **imports** *AlgebraFeedback FeedbacklessPerm*
begin
context *BaseOperationVars*
begin
thm *fb-perm*

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

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

8 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*: $is\text{-}lfp\ f\ x \Longrightarrow is\text{-}lfp\ f\ y \Longrightarrow x = y$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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*: $(\forall\ i\ .\ (a{::}nat \Rightarrow 'a\ option)\ i \le a\ (Suc\ i)) \Longrightarrow a\ n \ne \bot \Longrightarrow n \le i \Longrightarrow a\ i = a\ n$

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

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*: $(x \le y) = (fst\ x \le fst\ y \wedge snd\ x \le snd\ y)$
definition *less-prod-def*: $((x{::}'a{\times}'b) < y) = (x \le y \wedge \neg\ (y \le x))$
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*: $(\forall\ i{::}\ nat\ .\ (a{::}nat \Rightarrow 'a \times 'b)\ i \le a\ (Suc\ i)) \Longrightarrow \exists\ n\ .\ \forall\ i \ge n\ .\ a\ i = a\ n$
instance
end

end

9 Model of the HBD Algebra

theory *Model* **imports** *AlgebraFeedback 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* ≤ *u*) |
(*le-val* (*Bool v*) (*Bool u*)) = (*v* ≤ *u*) |
(*le-val* (*Nat v*) (*Nat u*)) = (*v* ≤ *u*) |
le-val - - = *False*

instantiation *Values* :: *order*
begin
definition *le-Values-def*: ((*v*::*Values*) ≤ *u*) = *le-val v u*
definition *less-Values-def*: ((*v*::*Values*) < *u*) = (*v* ≤ *u* ∧ ¬ *u* ≤ *v*)
instance
end

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

instantiation *list* :: (*order*) *order*
begin
definition *le-list-def*: ((*v*::*'a list*) ≤ *u*) = *le-list u v*
definition *less-list-def*: ((*v*::*'a list*) < *u*) = (*v* ≤ *u* ∧ ¬ *u* ≤ *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*) ⊆ *Some* ‘ {*v* . *tp v* = *x*})}

definition *has-in-out-type x y* = *has-in-type x* ∩ *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* ∈ *has-in-out-type x y* ⟹ *g* ∈ *has-in-out-type y z* ⟹ *g* $\circ_m$ *f* ∈ *has-in-out-type x z*

definition $TI\text{-}f\,f = (SOME\ x\ .\ (\exists\ y\ .\ f \in has\text{-}in\text{-}out\text{-}type\ x\ y))$

definition $TO\text{-}f\,f = (SOME\ y\ .\ (\exists\ x\ .\ f \in has\text{-}in\text{-}out\text{-}type\ x\ y))$

fun *pref* :: *Values list* ⇒ *Types list* ⇒ *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* ⇒ *Types list* ⇒ *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 $par\text{-}f\,f\,g\ v = (if\ tp\ v = (TI\text{-}f\,f)\ @\ (TI\text{-}f\ g)\ then\ Some\ (the\ (f\ (pref\ v\ (TI\text{-}f\,f)))\ @\ (the\ (g\ (suff\ v\ (TI\text{-}f\,f)))))\ else\ None)$

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

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-f x v* = (*if tp v* = *x then Some* [] *else None*)

lemma *Sink-has-in-out-type*: *Sink-f x* ∈ *has-in-out-type x* []

definition *Switch-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-f x y* ∈ *has-in-out-type* (*x* @ *y*) (*y* @ *x*)

primrec *fb-t* :: *Types* ⇒ (*Values* ⇒ *Values*) ⇒ *Values* **where**
fb-t int f = *Inte* (*Lfp* (λ *a* . *integer* (*f* (*Inte a*)))) |
fb-t bool f = *Bool* (*Lfp* (λ *a* . *boolean* (*f* (*Bool a*)))) |
fb-t nat f = *Nat* (*Lfp* (λ *a* . *natural* (*f* (*Nat a*))))

definition *fb-f f v* = (*if tp v* = *tl* (*TI-f f*) *then Some* (*tl* (*the* (*f* ((*fb-t* (*hd* (*TI-f f*)) (λ *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* ⟹ *mono B* ⟹ *mono C* ⟹ *mono-a f* ⟹ *mono-a* (λ*a b*. *C* (*f* (*A a*) (*B b*)))

lemma *fb-t-commute*: *mono-a f* ⟹ *mono-a g*
⟹ *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*: (⋀ *a* . *tv a* = *t* ⟹ *f a* = *g a*) ⟹ *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* ⟹ *f* ∈ *has-in-out-type x y* ⟹ *tp v* = *x*

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

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

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

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

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

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

lemma [*simp*]: $TI\text{-}f\ (ID\text{-}f\ ts) = ts$

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

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

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

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

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

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

thm *has-type-out-type*

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

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

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

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

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

thm *monoD*

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

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

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

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

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

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

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

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

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

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

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

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

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

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

typedef *func* = *typed-func*

definition *fb-func* f = *Abs-func* (*fb-f* (*Rep-func* f))

definition *TI-func* f = (*TI-f* (*Rep-func* f))
definition *TO-func* f = (*TO-f* (*Rep-func* f))
definition *ID-func* t = *Abs-func* (*ID-f* t)

definition *comp-func* $f\ g$ = *Abs-func* ((*Rep-func* g) $\circ_m$ (*Rep-func* f))

definition *parallel-func* $f\ g$ = *Abs-func* (*par-f* (*Rep-func* f) (*Rep-func* g))

definition *Dup-func* x = *Abs-func* (*Dup-f* x)

definition *Sink-func* x = *Abs-func* (*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* $y'\ z' \Longrightarrow$

$par\text{-}f\ g\ g' \circ_m par\text{-}f\ f\ f' = (par\text{-}f\ (g \circ_m f)\ (g' \circ_m f'))$

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

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

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

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

lemma [*simp*]: $TI\text{-}f\ (Sink\text{-}f\ ts) = ts$

lemma [*simp*]: $TO\text{-}f\ (Sink\text{-}f\ ts) = []$

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

lemma [*simp*]: $TI\text{-}f\ (Dup\text{-}f\ x) = x$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

10 Refinement Calculus.

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

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

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*)

term {. *x*, *z* . *P x y*.}

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

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

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

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

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

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*: $\{.p.\}\ o\ \{.p'.\} = \{.x\ .\ p\ x \wedge p'\ x.\}$

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

definition $trs\ r = \{.\ inpt\ r.\}\ o\ [:r:]$

lemma $trs\ (\lambda\ x\ y\ .\ x = y)\ q\ x = q\ x$

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

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

lemma $trs\ (\lambda\ x\ y\ .\ x \ge 0) \le trs\ (\lambda\ x\ y\ .\ x = y)$

lemma $trs\ (\lambda\ x\ y\ .\ x \ge 0)\ q\ x = (if\ q = \top\ then\ x \ge 0\ else\ False)$

lemma $[:r:] \sqcap [:r':] = [:r \sqcup r':]$

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 $[:(\lambda\ x\ y\ .\ p\ x \longrightarrow r\ x\ y) \sqcap ((\lambda\ x\ y\ .\ p'\ x \longrightarrow r'\ x\ y)):]$

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

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

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

term $INF\ b{:}X\ .\ A$

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 [*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* $[:r:] =$ *inpt* r

lemma $(S::'a::bot \Rightarrow 'b::boolean\text{-}algebra) \le S' \Longrightarrow$ *grd* $S' \le$ *grd* S

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

lemma [*simp*]: *inp* $(\lambda x\ y.\ p\ x \wedge r\ x\ y) = p \sqcap$ *inp* r

lemma [*simp*]: $p \le$ *inp* $r \Longrightarrow p \sqcap$ *inp* $r = p$

lemma *grd-spec*: *grd* $(\{.p.\}\ o\ [:r:]) = -p \sqcup$ *inp* 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 *mono* $(S::'a::\mathit{boolean\text{-}algebra} \Rightarrow 'b::\mathit{boolean\text{-}algebra}) \Longrightarrow (S = \mathit{Fail}) = (\mathit{fail}\ S = \top)$

lemma *Inf-not-eq*: $\mathit{Inf}\ \{X.\ \exists\, b.\ \neg\ x\ b \wedge (X = op \neq b)\} = x$

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 [*simp*]: $\{.\ \lambda\ (x,\ y)\ .\ True\ .\} = Skip$

lemma [*simp*]: $mono\ S \Longrightarrow mono\ S' \Longrightarrow mono\ (S\ o\ S')$

lemma *assert-true-skip-a*:$\{.\ \lambda\ x\ .\ True\ .\} = Skip$

lemma *assert-false-fail*: $\{.\bot::'a::boolean\text{-}algebra.\}\ = \bot$

lemma [*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*: $\{.p.\}\ o\ [:r:] = \{.p.\}\ o\ [:\lambda\ x\ y\ .\ p\ x \wedge r\ x\ y:]$

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 [*simp*]: $mono\ \{.p.\}$

lemma [*simp*]: $mono\ [.p.]$

lemma [*simp*]: $mono\ [:r:]$

lemma [*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 [*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 [*simp*]: $sconjunctive\ Fail$

thm *sconjunctive-simp*

lemma *sconjunctive-simp-c*: *sconjunctive* $(S::('a \Rightarrow bool) \Rightarrow 'b \Rightarrow bool) \Longrightarrow prec$ $S = \bot \Longrightarrow S = Fail$

thm *sconjunctive-simp*

lemma *demonic-eq-skip*: $[:\ op = :] = Skip$

definition $Havoc = [:\top:]$

definition $Magic = [:\bot::'a \Rightarrow 'b::boolean\text{-}algebra:]$

lemma *Magic-top*: $Magic = \top$

lemma [*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 [*simp*]: $[:r:]\ o\ Magic = Magic$

lemma [*simp*]: $Magic\ o\ S = Magic$

lemma [*simp*]: $Havoc \circ Magic = Magic$

lemma $Havoc\ \top = \top$

lemma [*simp*]: $Skip\ p = p$

lemma *demonic-pair-skip*: $[:\ \lambda(x,\ y)\ (u,\ v).\ x = u \wedge y = v\ :] = Skip$

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 [*simp*]: $prec\ Skip = (\top::'a{\Rightarrow}bool)$

definition *update* :: $('a \Rightarrow 'b) \Rightarrow ('b \Rightarrow bool) \Rightarrow 'a \Rightarrow bool$ $([-\text{-}-])$ **where**
$[-f-] = [:x \leadsto y\ .\ y = f\ x:]$
syntax

-update :: *patterns* => *logic* => *logic* ((*1*[−-./ -−]))

translations

-update (*-patterns* *x* *xs*) *F* == *CONST update* (*CONST id* (*-abs* (*-pattern* *x* *xs*) *F*))

-update *x* *F* == *CONST update* (*CONST id* (*-abs* *x* *F*))

term $[-x,y\ .\ (x+y,\ y-x)-]$

term $[-x,y\ .\ x+y-]$

lemma *update-o-def*: $[-f\ o\ g-] = [-(\lambda\ x\ .\ f\ (g\ x))-]$

lemma *update-assert-comp*: $[-f-]\ o\ \{.p.\} = \{.p\ o\ f.\}\ o\ [-f-]$

lemma *update-comp*: $[-f-]\ o\ [-g-] = [-g\ o\ f-]$

lemma *convert*: $(\lambda\ x\ y\ .\ (S::('a \Rightarrow bool) \Rightarrow ('b \Rightarrow bool))\ x\ (f\ y)) = [-f-]\ o\ S$

lemma *update-id-Skip*: $[-id-] = Skip$

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 *assert-fail*: $\{.p::'a::boolean\text{-}algebra.\}\ o\ \bot = \bot$

lemma *angelic-assert*: $\{:r:\}\ o\ \{.p.\} = \{:x \rightsquigarrow y\ .\ r\ x\ y \wedge p\ y:\}$

lemma *prec-rel-eq*: $p = p' \Longrightarrow r = r' \Longrightarrow \{.p.\}\ o\ [:r:] = \{.p'.\}\ o\ [:r':]$

lemma *prec-rel-le*: $p \leq p' \Longrightarrow (\bigwedge\ x\ .\ p\ x \Longrightarrow r'\ x \leq r\ x) \Longrightarrow \{.p.\}\ o\ [:r:] \leq \{.p'.\}\ o\ [:r':]$

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 *sconjunctiveE*: *sconjunctive* $S \Longrightarrow (\exists\ p\ r\ .\ S = \{.\ p\ .\}\ o\ [:\ r :: 'a \Rightarrow 'b \Rightarrow bool:])$

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

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

lemma *nonmagic-assert*: *non-magic* $\{.p::'a::boolean\text{-}algebra.\}$

end

11 Hoare Total Correctness Rules.

theory *Hoare* **imports** *Refinement*
begin

definition *if-stm* $p\ S\ T = ([.p.]\ o\ S) \sqcap ([.-p.]\ o\ T)$
definition *while-stm* $p\ S = lfp\ (\lambda\ X\ .\ if\text{-}stm\ p\ (S\ o\ X)\ Skip)$
definition *Hoare* $p\ S\ q = (p \le S\ q)$

definition *Sup-less* $x\ (w::'b::wellorder) = Sup\ \{y :: 'a::complete\text{-}lattice\ .\ \exists\ v < w\ .\ y = x\ v\}$

lemma *Sup-less-upper*:
$v < w \Longrightarrow P\ v \le Sup\text{-}less\ P\ w$

lemma *Sup-less-least*:
$(!!\ v\ .\ v < w \Longrightarrow P\ v \le Q) \Longrightarrow Sup\text{-}less\ P\ w \le Q$

theorem *fp-wf-induction*:

$f\ x\ = x \Longrightarrow$ *mono* $f \Longrightarrow (\forall\ w\ .\ (y\ w) \le f\ (\textit{Sup-less}\ y\ w)) \Longrightarrow \textit{Sup}\ (\textit{range}\ y) \le x$

theorem *lfp-wf-induction*: *mono* $f \Longrightarrow (\forall\ w\ .\ (p\ w) \le f\ (\textit{Sup-less}\ p\ w)) \Longrightarrow \textit{Sup}\ (\textit{range}\ p) \le \textit{lfp}\ f$

theorem *lfp-wf-induction-a*: *mono* $f \Longrightarrow (\forall\ w\ .\ (p\ w) \le f\ (\textit{Sup-less}\ p\ w)) \Longrightarrow (\textit{SUP}\ a.\ p\ a) \le \textit{lfp}\ f$

theorem *lfp-wf-induction-b*: *mono* $f \Longrightarrow (\forall\ w\ .\ (p\ w) \le f\ (\textit{Sup-less}\ p\ w)) \Longrightarrow S \le (\textit{SUP}\ a.\ p\ a) \Longrightarrow S \le \textit{lfp}\ f$

lemma [*simp*]: *mono* $S \Longrightarrow$ *mono* $(\lambda X.\ \textit{if-stm}\ b\ (S \circ X)\ T)$

lemma [*simp*]: $\textit{Sup-less}\ (\lambda n\ x.\ t\ x = n)\ n = (\lambda\ x\ .\ (t\ x < n))$

lemma [*simp*]: $(\textit{SUP}\ a.\ \{.x\ .t\ x = a.\} \circ S) = S$
definition $\textit{mono-mono}\ F = (\textit{mono}\ F \wedge (\forall\ f\ .\ \textit{mono}\ f \longrightarrow \textit{mono}\ (F\ f)))$

definition $\textit{post-fun}\ (p{::}'a{::}\textit{order})\ q = (\textit{if}\ p \le q\ \textit{then}\ \top\ \textit{else}\ \bot)$

lemma *post-mono* [*simp*]: $\textit{mono}\ (\textit{post-fun}\ p :: (\text{-}{::}\{\textit{order-bot},\textit{order-top}\}))$

lemma *post-refin* [*simp*]: *mono* $S \Longrightarrow ((S\ p){::}'a{::}\textit{bounded-lattice}) \sqcap (\textit{post-fun}\ p)\ x \le S\ x$

lemma *post-top* [*simp*]: $\textit{post-fun}\ p\ p = \top$

theorem *hoare-refinement-post*:
mono $f \Longrightarrow\ (\textit{Hoare}\ x\ \ f\ y) = (\{.x{::}'a{::}\textit{boolean-algebra}.\}\ o\ (\textit{post-fun}\ y) \le f)$

lemma *assert-Sup-range*: $\{.\textit{Sup}\ (\textit{range}\ (p{::}'W \Rightarrow 'a{::}\textit{complete-distrib-lattice})).\} = \textit{Sup}(\textit{range}\ (\textit{assert}\ o\ p))$

lemma *Sup-range-comp*: $(\textit{Sup}\ (\textit{range}\ p))\ o\ S = \textit{Sup}\ (\textit{range}\ (\lambda\ w\ .\ ((p\ w)\ o\ S)))$

theorem *lfp-mono* [*simp*]:
$\textit{mono-mono}\ F \Longrightarrow \textit{mono}\ (\textit{lfp}\ F)$

lemma *Sup-less-comp*: $(\textit{Sup-less}\ P)\ w\ o\ S = \textit{Sup-less}\ (\lambda\ w\ .\ ((P\ w)\ o\ S))\ w$

lemma *assert-Sup*: $\{.\textit{Sup}\ (X{::}'a{::}\textit{complete-distrib-lattice set}).\} = \textit{Sup}\ (\textit{assert}\ `\ X)$

lemma *Sup-less-assert*: $\textit{Sup-less}\ (\lambda w.\ \{.\ (p\ w){::}'a{::}\textit{complete-distrib-lattice}\ .\})\ w = \{.\textit{Sup-less}\ p\ w.\}$

lemma [*simp*]: $\textit{Sup-less}\ (\lambda n.\ \{.x.\ t\ x = n.\} \circ S)\ n = \{.x.\ t\ x < n.\} \circ S$

theorem *hoare-fixpoint*:
$mono\text{-}mono\ F \Longrightarrow$
$(\forall\ f\ w\ .\ mono\ f \longrightarrow (Hoare\ (Sup\text{-}less\ p\ w)\ f\ y \longrightarrow Hoare\ ((p\ w)::'a \Rightarrow bool)\ (F\ f)\ y)) \Longrightarrow Hoare(Sup\ (range\ p))\ (lfp\ F)\ y$

lemma *if-mono*[*simp*]: $mono\ S \Longrightarrow mono\ T \Longrightarrow mono\ (if\text{-}stm\ b\ S\ T)$

lemma [*simp*]: $mono\ x \Longrightarrow mono\text{-}mono\ (\lambda X\ .\ if\text{-}stm\ b\ (x \circ X)\ Skip)$

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*:
$(Hoare\ P\ [.R.]\ Q) = (P \sqcap R \leq Q)$

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 *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*: $mono\ S \Longrightarrow p \leq p' \Longrightarrow q' \leq q \Longrightarrow Hoare\ p'\ S\ q' \Longrightarrow Hoare\ p\ S\ q$

lemma [*simp*]: $mono\ x \Longrightarrow mono\ (while\text{-}stm\ b\ x)$

lemma *hoare-while-a*:
$mono\ x \Longrightarrow (\forall\ w\ .\ Hoare\ ((p\ w) \sqcap b)\ x\ (Sup\text{-}less\ p\ w)) \Longrightarrow p' \leq (Sup\ (range\ p)) \Longrightarrow ((Sup\ (range\ p)) \sqcap -b) \leq q$
$\Longrightarrow Hoare\ p'\ (while\text{-}stm\ b\ x)\ q$

lemma *hoare-update*: $p \leq q\ o\ f \Longrightarrow Hoare\ p\ [-f-]\ q$

lemma *hoare-demonic*: $(\bigwedge\ x\ y\ .\ p\ x \Longrightarrow r\ x\ y \Longrightarrow q\ y) \Longrightarrow Hoare\ p\ [:r:]\ q$

lemma *refinement-hoare*: $S \leq T \Longrightarrow Hoare\ (p::'a::order)\ S\ (q) \Longrightarrow Hoare\ p\ T\ q$

lemma *refinement-hoare-iff*: $(S \leq T) = (\forall\ p\ q\ .\ Hoare\ (p::'a::order)\ S\ (q) \longrightarrow Hoare\ p\ T\ q)$

end

12 Nondeterministic Algorithm.

theory *AlgorithmFeedback* **imports** *Feedback Hoare*
begin

context *BaseOperationVars*
begin

definition *TranslateHBD* =
while-stm (λ *As* . *length As* > *1*)(
[:*As* $\leadsto$ *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*) $\wedge$ *As'* = (*FB* (*FB A* ;; *FB B*)) # *Bs*:]
)
o [−(λ *As* . *FB*(*As* ! *0*))−]

lemma [*simp*]:*Suc 0* $\leq$ *length As-init* $\Longrightarrow$
Hoare (λ*As*. *in-out-equiv* (*FB* (*As* ! *0*)) (*FB* (*Parallel-list As-init*))) [−λ*As*. *FB* (*As* ! *0*)−] (λ*S*. *in-out-equiv S* (*FB* (*Parallel-list As-init*)))

definition *invariant As-init n As* = (*length As* = *n* $\wedge$ *io-distinct As* $\wedge$ *in-out-equiv* (*FB* (*Parallel-list As*)) (*FB* (*Parallel-list As-init*)) $\wedge$ *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* [] *op* $\oplus$ (*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*]: $io\text{-}distinct\ x \Longrightarrow perm\ x\ (A\ \#\ B\ \#\ Bs) \Longrightarrow set\ (Out\ A) \cap set\ (Out\ B) = \{\}$

lemma *distinct-Par-equiv-b*: $io\text{-}distinct\ As \Longrightarrow perm\ As\ (Bs\ @\ Cs) \Longrightarrow in\text{-}out\text{-}equiv\ (FB\ (FB\ (Parallel\text{-}list\ Bs)\ |||\ Parallel\text{-}list\ Cs))\ (FB\ (Parallel\text{-}list\ As))$

lemma *distinct-Par-equiv*: $io\text{-}distinct\ As\text{-}init \Longrightarrow Suc\ 0 \leq length\ As\text{-}init \Longrightarrow length\ As = w \Longrightarrow io\text{-}distinct\ As \Longrightarrow in\text{-}out\text{-}equiv\ (FB\ (Parallel\text{-}list\ As))\ (FB\ (Parallel\text{-}list\ As\text{-}init)) \Longrightarrow Suc\ 0 < w \Longrightarrow Suc\ 0 < length\ Bs \Longrightarrow perm\ As\ (Bs\ @\ Cs) \Longrightarrow io\text{-}distinct\ (FB\ (Parallel\text{-}list\ Bs)\ \#\ Cs) \wedge in\text{-}out\text{-}equiv\ (FB\ (FB\ (Parallel\text{-}list\ Bs)\ |||\ Parallel\text{-}list\ Cs))\ (FB\ (Parallel\text{-}list\ As\text{-}init))$

lemma *AAAA-x*[*simp*]: $io\text{-}distinct\ As\text{-}init \Longrightarrow Suc\ 0 \leq length\ As\text{-}init \Longrightarrow invariant\ As\text{-}init\ w\ x \Longrightarrow Suc\ 0 < length\ x \Longrightarrow Suc\ 0 < length\ Bs \Longrightarrow perm\ x\ (Bs\ @\ Cs) \Longrightarrow invariant\ As\text{-}init\ (Suc\ (length\ Cs))\ (FB\ (Parallel\text{-}list\ Bs)\ \#\ Cs)$

term $\{1,2,3\} - \{2,3\}$

thm *ParallelId-right*

lemma [*simp*]: $io\text{-}distinct\ As\text{-}init \Longrightarrow Suc\ 0 \leq length\ As\text{-}init \Longrightarrow invariant\ As\text{-}init\ w\ x \Longrightarrow Suc\ 0 < length\ x \Longrightarrow perm\ x\ (A\ \#\ B\ \#\ Bs) \Longrightarrow invariant\ As\text{-}init\ (Suc\ (length\ Bs))\ (FB\ (FB\ A\ ;;\ FB\ B)\ \#\ Bs)$

lemma [*simp*]: $io\text{-}distinct\ As\text{-}init \Longrightarrow Suc\ 0 \leq length\ As\text{-}init \Longrightarrow Hoare\ (invariant\ As\text{-}init\ w \sqcap (\lambda As.\ Suc\ 0 < length\ As))\ [:As \rightsquigarrow As'.\exists\, Bs.\ Suc\ 0 < length\ Bs \wedge (\exists\, Cs.\ perm\ As\ (Bs\ @\ Cs) \wedge As' = FB\ (Parallel\text{-}list\ Bs)\ \#\ Cs):]\ (Sup\text{-}less\ (invariant\ As\text{-}init)\ w)$

lemma [*simp*]: $io\text{-}distinct\ As\text{-}init \Longrightarrow Suc\ 0 \leq length\ As\text{-}init \Longrightarrow Hoare\ (invariant\ As\text{-}init\ w \sqcap (\lambda As.\ Suc\ 0 < length\ As))\ [:As \rightsquigarrow As'.\exists\, A\ B\ Bs.\ perm\ As\ (A\ \#\ B\ \#\ Bs) \wedge As' = FB\ (FB\ A\ ;;\ FB\ B)\ \#\ Bs:]\ (Sup\text{-}less\ (invariant\ As\text{-}init)\ w)$

theorem *CorrectnessTranslateHBD*: $io\text{-}distinct\ As\text{-}init \Longrightarrow length\ As\text{-}init \geq 1 \Longrightarrow Hoare\ (io\text{-}distinct \sqcap (\lambda\ As\ .\ As = As\text{-}init))\ TranslateHBD\ (\lambda\ S\ .\ in\text{-}out\text{-}equiv\ S\ (FB\ (Parallel\text{-}list\ As\text{-}init)))$
end

end

13 Feedbackless Algorithm

theory *AlgorithmFeedbackless* **imports** *FeedbacklessPerm Hoare*

begin
context *BaseOperationFeedbacklessVars*
begin
definition *WhileFeedbackless* =
while-stm (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*]):]

definition *TranslateHBDFeedbackless* = *WhileFeedbackless o* [−(λ *As* . *Parallel-list As*)−]

definition *ok-fbless As* = (*Deterministic As* $\wedge$ *loop-free As* $\wedge$ *Type-OK As*)

definition *TranslateHBDRec* = {. *ok-fbless* .}
o [:*As* $\rightsquigarrow$ *As'* . $\exists$ *L* . *perm* (*VarFB* (*Parallel-list As*)) *L* $\wedge$ *As'* = *fb-less L As* :]

lemma [*simp*]:{.*As*. *length* (*VarFB* (*Parallel-list As*)) = *w*.} (*TranslateHBDRec x*) *y* $\Longrightarrow$ [. − (λ*As*. *internal As* $\neq$ {}) .] *x y*

lemma *internal-fb-less-step*: *loop-free As* $\Longrightarrow$ *Type-OK As* $\Longrightarrow$ *A* $\in$ *set As* $\Longrightarrow$ *out A* $\in$ *internal As* $\Longrightarrow$ *internal* (*fb-less-step A* (*As* $\ominus$ [*A*])) = *internal As* − {*out A*}

lemma *ok-fbless-fb-less-step*: *ok-fbless As* $\Longrightarrow$ *A* $\in$ *set As* $\Longrightarrow$ *out A* $\in$ *internal As* $\Longrightarrow$ *ok-fbless* (*fb-less-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* $[-(\lambda$ *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* $[-(\lambda$ *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