Constructive Cryptography in HOL: the Communication Modeling Aspect

Andreas Lochbihler and S. Reza Sefidgar

June 17, 2024

Abstract

Constructive Cryptography (CC) [8, 7, 9] introduces an abstract approach to composable security statements that allows one to focus on a particular aspect of security proofs at a time. Instead of proving the properties of concrete systems, CC studies system classes, i.e., the shared behavior of similar systems, and their transformations.

Modeling of systems communication plays a crucial role in composability and reusability of security statements; yet, this aspect has not been studied in any of the existing CC results. We extend our previous CC formalization [5, 6] with a new semantic domain called Fused Resource Templates (FRT) that abstracts over the systems communication patterns in CC proofs. This widens the scope of cryptography proof formalizations in the CryptHOL library [4, 3, 2].

This formalization is described in [1].

Contents

theory *More-CC* **imports**
Constructive-Cryptography.Constructive-Cryptography
begin

1 Material for Isabelle library

lemma *eq-alt-conversep*: $(=) = (\mathit{BNF\text{-}Def.Grp}\ \mathit{UNIV}\ \mathit{id})^{-1-1}$
⟨*proof*⟩

parametric-constant
swap-parametric [*transfer-rule*]: *prod.swap-def*

lemma *Sigma-parametric* [*transfer-rule*]: **includes** *lifting-syntax* **shows**
$(\mathit{rel\text{-}set}\ A ===> (A ===> \mathit{rel\text{-}set}\ B) ===> \mathit{rel\text{-}set}\ (\mathit{rel\text{-}prod}\ A\ B))\ \mathit{Sigma}\ \mathit{Sigma}$
⟨*proof*⟩

lemma *empty-eq-Plus* [*simp*]: $\{\} = A <+> B \longleftrightarrow A = \{\} \wedge B = \{\}$
⟨*proof*⟩

lemma *insert-Inl-Plus* [*simp*]: $\mathit{insert}\ (\mathit{Inl}\ x)\ (A <+> B) = \mathit{insert}\ x\ A <+> B$
⟨*proof*⟩

lemma *insert-Inr-Plus* [*simp*]: $\mathit{insert}\ (\mathit{Inr}\ x)\ (A <+> B) = A <+> \mathit{insert}\ x\ B$
⟨*proof*⟩

lemma *map-sum-image-Plus* [*simp*]: $\mathit{map\text{-}sum}\ f\ g\ `\ (A <+> B) = f\ `\ A <+> g\ `\ B$
⟨*proof*⟩

lemma *Plus-subset-Plus-iff* [*simp*]: $A <+> B \subseteq C <+> D \longleftrightarrow A \subseteq C \wedge B \subseteq D$
⟨*proof*⟩

lemma *map-sum-eq-Inl-iff*: $\mathit{map\text{-}sum}\ f\ g\ x = \mathit{Inl}\ y \longleftrightarrow (\exists x'.\ x = \mathit{Inl}\ x' \wedge y = f\ x')$
⟨*proof*⟩

lemma *map-sum-eq-Inr-iff*: $\mathit{map\text{-}sum}\ f\ g\ x = \mathit{Inr}\ y \longleftrightarrow (\exists x'.\ x = \mathit{Inr}\ x' \wedge y = g\ x')$
⟨*proof*⟩

lemma *surj-map-sum*: $\mathit{surj}\ (\mathit{map\text{-}sum}\ f\ g)$ **if** $\mathit{surj}\ f\ \mathit{surj}\ g$
⟨*proof*⟩

lemma *bij-map-sumI* [*simp*]: $\mathit{bij}\ (\mathit{map\text{-}sum}\ f\ g)$ **if** $\mathit{bij}\ f\ \mathit{bij}\ g$
⟨*proof*⟩

lemma *inv-map-sum* [*simp*]:
$[\![\ \mathit{bij}\ f;\ \mathit{bij}\ g\ ]\!] \Longrightarrow \mathit{inv\text{-}into}\ \mathit{UNIV}\ (\mathit{map\text{-}sum}\ f\ g) = \mathit{map\text{-}sum}\ (\mathit{inv\text{-}into}\ \mathit{UNIV}\ f)$

(*inv-into UNIV g*)
⟨*proof*⟩

context *conditionally-complete-lattice* **begin**

lemma *admissible-le1I*:
ccpo.admissible lub ord (λ*x*. *f x* ≤ *y*)
if *cont lub ord Sup* (≤) *f*
⟨*proof*⟩

lemma *admissible-le1-mcont* [*cont-intro*]:
ccpo.admissible lub ord (λ*x*. *f x* ≤ *y*) **if** *mcont lub ord Sup* (≤) *f*
⟨*proof*⟩

end

lemma *eq-alt-conversep2*: (=) = ((*BNF-Def.Grp UNIV id*)$^{-1-1}$)$^{-1-1}$
⟨*proof*⟩

lemma *nn-integral-indicator-singleton1* [*simp*]:
assumes [*measurable*]: {*y*} ∈ *sets M*
shows (∫$^{+}$*x*. *indicator* {*y*} *x* ∗ *f x* ∂*M*) = *emeasure M* {*y*} ∗ *f y*
⟨*proof*⟩

lemma *nn-integral-indicator-singleton1′* [*simp*]:
assumes {*y*} ∈ *sets M*
shows (∫$^{+}$*x*. *indicator* {*x*} *y* ∗ *f x* ∂*M*) = *emeasure M* {*y*} ∗ *f y*
⟨*proof*⟩

1.1 Probabilities

lemma *pmf-eq-1-iff*: *pmf p x* = *1* ⟷ *p* = *return-pmf x* (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *measure-spmf-cong*: *measure* (*measure-spmf p*) *A* = *measure* (*measure-spmf p*) *B*
if *A* ∩ *set-spmf p* = *B* ∩ *set-spmf p*
⟨*proof*⟩

definition *weight-spmf′* **where** *weight-spmf′* = *weight-spmf*
lemma *weight-spmf′-parametric* [*transfer-rule*]: *rel-fun* (*rel-spmf A*) (=) *weight-spmf′ weight-spmf′*
⟨*proof*⟩

lemma *bind-spmf-to-nat-on*:
bind-spmf (*map-spmf* (*to-nat-on* (*set-spmf p*)) *p*) (λ*n*. *f* (*from-nat-into* (*set-spmf p*) *n*)) = *bind-spmf p f*
⟨*proof*⟩

lemma *try-cond-spmf-fst*:
try-spmf (*cond-spmf-fst p x*) *q* = (*if x* ∈ *fst* ‘ *set-spmf p then cond-spmf-fst p x else q*)
⟨*proof*⟩

lemma *measure-try-spmf*:
measure (*measure-spmf* (*try-spmf p q*)) *A* = *measure* (*measure-spmf p*) *A* + *pmf p None* ∗ *measure* (*measure-spmf q*) *A*
⟨*proof*⟩

lemma *rel-spmf-OO-trans-strong*:
⟦ *rel-spmf R p q*; *rel-spmf S q r* ⟧ ⟹ *rel-spmf* (*R OO eq-onp* (λ*x*. *x* ∈ *set-spmf q*) *OO S*) *p r*
⟨*proof*⟩

lemma *mcont2mcont-spmf* [*cont-intro*]:
mcont lub ord Sup (≤) (λ*p*. *spmf* (*f p*) *x*)
if *mcont lub ord lub-spmf* (*ord-spmf* (=)) *f*
⟨*proof*⟩

lemma *ord-spmf-try-spmf2*: *ord-spmf R p* (*try-spmf p q*) **if** *rel-spmf R p p*
⟨*proof*⟩

lemma *ord-spmf-lossless-spmfD1*:
assumes *ord-spmf R p q*
and *lossless-spmf p*
shows *rel-spmf R p q*
⟨*proof*⟩

lemma *restrict-spmf-mono*:
ord-spmf (=) *p q* ⟹ *ord-spmf* (=) (*p* ↾ *A*) (*q* ↾ *A*)
⟨*proof*⟩

lemma *restrict-lub-spmf*:
assumes *chain*: *Complete-Partial-Order.chain* (*ord-spmf* (=)) *Y*
shows *restrict-spmf* (*lub-spmf Y*) *A* = *lub-spmf* ((λ*p*. *restrict-spmf p A*) ‘ *Y*) (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *mono2mono-restrict-spmf* [*THEN spmf.mono2mono*]:
shows *monotone-restrict-spmf*: *monotone* (*ord-spmf* (=)) (*ord-spmf* (=)) (λ*p*. *p* ↾ *A*)
⟨*proof*⟩

lemma *mcont2mcont-restrict-spmf* [*THEN spmf.mcont2mcont*, *cont-intro*]:
shows *mcont-restrict-spmf*: *mcont lub-spmf* (*ord-spmf* (=)) *lub-spmf* (*ord-spmf* (=)) (λ*p*. *restrict-spmf p A*)
⟨*proof*⟩

lemma *ord-spmf-case-option*: *ord-spmf R* (*case x of None* ⇒ *a* | *Some y* ⇒ *b y*) (*case x of None* ⇒ *a'* | *Some y* ⇒ *b' y*)
if *ord-spmf R a a'* ⋀*y*. *ord-spmf R* (*b y*) (*b' y*) ⟨*proof*⟩

lemma *ord-spmf-map-spmfI*: *ord-spmf* (=) (*map-spmf f p*) (*map-spmf f q*) **if** *ord-spmf* (=) *p q*
⟨*proof*⟩

1.1.1 Conditional probabilities

lemma *mk-lossless-cond-spmf* [*simp*]: *mk-lossless* (*cond-spmf p A*) = *cond-spmf p A*
⟨*proof*⟩

context
fixes *p* :: *'a pmf*
and *f* :: *'a* ⇒ *'b pmf*
and *A* :: *'b set*
and *F* :: *'a* ⇒ *real*
defines *F* ≡ λ*x*. *pmf p x* ∗ *measure* (*measure-pmf* (*f x*)) *A* / *measure* (*measure-pmf* (*bind-pmf p f*)) *A*
begin

definition *cond-bind-pmf* :: *'a pmf* **where** *cond-bind-pmf* = *embed-pmf F*

lemma *cond-bind-pmf-nonneg*: *F x* ≥ *0*
⟨*proof*⟩

context assumes *defined*: *A* ∩ (⋃*x*∈*set-pmf p*. *set-pmf* (*f x*)) ≠ {} **begin**

private lemma *nonzero*: *measure* (*measure-pmf* (*bind-pmf p f*)) *A* > *0*
⟨*proof*⟩

lemma *cond-bind-pmf-prob*: (∫⁺ *x*. *F x* ∂*count-space UNIV*) = *1*
⟨*proof*⟩

lemma *pmf-cond-bind-pmf*: *pmf cond-bind-pmf x* = *F x*
⟨*proof*⟩

lemma *set-cond-bind-pmf*: *set-pmf cond-bind-pmf* = {*x*∈*set-pmf p*. *set-pmf* (*f x*) ∩ *A* ≠ {}}
⟨*proof*⟩

lemma *cond-bind-pmf*: *cond-pmf* (*bind-pmf p f*) *A* = *bind-pmf cond-bind-pmf* (λ*x*. *cond-pmf* (*f x*) *A*)
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

end

end

lemma *cond-spmf-try1*:
cond-spmf (*try-spmf p q*) *A* = *cond-spmf p A* **if** *set-spmf q* ∩ *A* = {}
⟨*proof*⟩

lemma *cond-spmf-cong*: *cond-spmf p A* = *cond-spmf p B* **if** *A* ∩ *set-spmf p* = *B* ∩ *set-spmf p*
⟨*proof*⟩

lemma *cond-spmf-pair-spmf*:
cond-spmf (*pair-spmf p q*) (*A* × *B*) = *pair-spmf* (*cond-spmf p A*) (*cond-spmf q B*) (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *cond-spmf-pair-spmf1*:
cond-spmf-fst (*map-spmf* (λ((*x*, *s'*), *y*). (*f x*, *s'*, *y*)) (*pair-spmf p q*)) *x* =
pair-spmf (*cond-spmf-fst* (*map-spmf* (λ(*x*, *s'*). (*f x*, *s'*)) *p*) *x*) *q* (**is** *?lhs* = *?rhs*)
if *lossless-spmf q*
⟨*proof*⟩

lemma *try-cond-spmf*: *try-spmf* (*cond-spmf p A*) *q* = (*if set-spmf p* ∩ *A* ≠ {} *then cond-spmf p A else q*)
⟨*proof*⟩

lemma *cond-spmf-try2*:
cond-spmf (*try-spmf p q*) *A* = (*if lossless-spmf p then return-pmf None else cond-spmf q A*) **if** *set-spmf p* ∩ *A* = {}
⟨*proof*⟩

definition *cond-bind-spmf* :: *'a spmf* ⇒ (*'a* ⇒ *'b spmf*) ⇒ *'b set* ⇒ *'a spmf* **where**

cond-bind-spmf p f A =
(*if* ∃ *x*∈*set-spmf p*. *set-spmf* (*f x*) ∩ *A* ≠ {} *then*
cond-bind-pmf p (λ*x*. *case x of None* ⇒ *return-pmf None* | *Some x* ⇒ *f x*) (*Some* ‘ *A*)
else return-pmf None)

context begin

private lemma *defined*: ⟦ *y* ∈ *set-spmf* (*f x*); *y* ∈ *A*; *x* ∈ *set-spmf p* ⟧
⟹ *Some* ‘ *A* ∩ (⋃ *x*∈*set-pmf p*. *set-pmf* (*case x of None* ⇒ *return-pmf None* | *Some x* ⇒ *f x*)) ≠ {}
⟨*proof*⟩

lemma *spmf-cond-bind-spmf* [*simp*]:
spmf (*cond-bind-spmf p f A*) *x* = *spmf p x* ∗ *measure* (*measure-spmf* (*f x*)) *A* / *measure* (*measure-spmf* (*bind-spmf p f*)) *A*
⟨*proof*⟩

lemma *set-cond-bind-spmf* [*simp*]:
set-spmf (*cond-bind-spmf p f A*) = {*x*∈*set-spmf p*. *set-spmf* (*f x*) ∩ *A* ≠ {}}
⟨*proof*⟩

lemma *cond-bind-spmf*: *cond-spmf* (*bind-spmf p f*) *A* = *bind-spmf* (*cond-bind-spmf p f A*) (λ*x*. *cond-spmf* (*f x*) *A*)
⟨*proof*⟩

end

lemma *cond-spmf-fst-parametric* [*transfer-rule*]: **includes** *lifting-syntax* **shows**
(*rel-spmf* (*rel-prod* (=) *B*) ===> (=) ===> *rel-spmf B*) *cond-spmf-fst cond-spmf-fst*
⟨*proof*⟩

lemma *cond-spmf-fst-map-prod*:
cond-spmf-fst (*map-spmf* (λ(*x*, *y*). (*f x*, *g x y*)) *p*) (*f x*) = *map-spmf* (*g x*) (*cond-spmf-fst p x*)
if *inj-on f* (*insert x* (*fst* ‘ *set-spmf p*))
⟨*proof*⟩

lemma *cond-spmf-fst-map-prod-inj*:
cond-spmf-fst (*map-spmf* (λ(*x*, *y*). (*f x*, *g x y*)) *p*) (*f x*) = *map-spmf* (*g x*) (*cond-spmf-fst p x*)
if *inj f*
⟨*proof*⟩

definition *cond-bind-spmf-fst* :: *'a spmf* ⇒ (*'a* ⇒ *'b spmf*) ⇒ *'b* ⇒ *'a spmf* **where**
cond-bind-spmf-fst p f x = *cond-bind-spmf p* (*map-spmf* (λ*b*. (*b*, ())) ∘ *f*) ({*x*} × *UNIV*)

lemma *cond-bind-spmf-fst-map-spmf-fst*:
cond-bind-spmf-fst p (*map-spmf fst* ∘ *f*) *x* = *cond-bind-spmf p f* ({*x*} × *UNIV*) (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *cond-spmf-fst-bind*: *cond-spmf-fst* (*bind-spmf p f*) *x* =
bind-spmf (*cond-bind-spmf-fst p* (*map-spmf fst* ∘ *f*) *x*) (λ*y*. *cond-spmf-fst* (*f y*) *x*)
⟨*proof*⟩

lemma *spmf-cond-bind-spmf-fst* [*simp*]:
spmf (*cond-bind-spmf-fst p f x*) *i* = *spmf p i* ∗ *spmf* (*f i*) *x* / *spmf* (*bind-spmf p f*) *x*
⟨*proof*⟩

lemma *set-cond-bind-spmf-fst* [*simp*]:
set-spmf (*cond-bind-spmf-fst p f x*) = {*y* ∈ *set-spmf p*. *x* ∈ *set-spmf* (*f y*)}
⟨*proof*⟩

lemma *map-cond-spmf-fst*: *map-spmf f* (*cond-spmf-fst p x*) = *cond-spmf-fst* (*map-spmf* (*apsnd f*) *p*) *x*
⟨*proof*⟩

lemma *cond-spmf-fst-try1*:
cond-spmf-fst (*try-spmf p q*) *x* = *cond-spmf-fst p x* **if** *x* ∉ *fst* ‘ *set-spmf q*
⟨*proof*⟩

lemma *cond-spmf-fst-try2*:
cond-spmf-fst (*try-spmf p q*) *x* = (*if lossless-spmf p then return-pmf None else cond-spmf-fst q x*) **if** *x* ∉ *fst* ‘ *set-spmf p*
⟨*proof*⟩

lemma *cond-spmf-fst-map-inj*:
cond-spmf-fst (*map-spmf* (*apfst f*) *p*) (*f x*) = *cond-spmf-fst p x* **if** *inj f*
⟨*proof*⟩

lemma *cond-spmf-fst-pair-spmf1*:
cond-spmf-fst (*map-spmf* (λ(*x*, *y*). (*f x*, *g x y*)) (*pair-spmf p q*)) *a* =
bind-spmf (*cond-spmf-fst* (*map-spmf* (λ*x*. (*f x*, *x*)) *p*) *a*) (λ*x*. *map-spmf* (*g x*) (*mk-lossless q*)) (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *cond-spmf-fst-return-spmf'*:
cond-spmf-fst (*return-spmf* (*x*, *y*)) *z* = (*if x* = *z then return-spmf y else return-pmf None*)
⟨*proof*⟩

2 Material for CryptHOL

lemma *left-gpv-lift-spmf* [*simp*]: *left-gpv* (*lift-spmf p*) = *lift-spmf p*
⟨*proof*⟩

lemma *right-gpv-lift-spmf* [*simp*]: *right-gpv* (*lift-spmf p*) = *lift-spmf p*
⟨*proof*⟩

lemma *map'-lift-spmf*: *map-gpv' f g h* (*lift-spmf p*) = *lift-spmf* (*map-spmf f p*)
⟨*proof*⟩

lemma *in-set-sample-uniform* [*simp*]: *x* ∈ *set-spmf* (*sample-uniform n*) ⟷ *x* < *n*
⟨*proof*⟩

lemma (**in** *cyclic-group*) *inj-on-generator-iff* [*simp*]: ⟦ *x* < *order G*; *y* < *order G* ⟧ ⟹ **g** [^] *x* = **g** [^] *y* ⟷ *x* = *y*

⟨*proof*⟩

lemma *map-ℐ-bot* [*simp*]: *map-ℐ f g* ⊥ = ⊥
⟨*proof*⟩

lemma *map-ℐ-Inr-plus* [*simp*]: *map-ℐ Inr f* (*ℐ1* ⊕$_ℐ$ *ℐ2*) = *map-ℐ id* (*f* ∘ *Inr*) *ℐ2*
⟨*proof*⟩

lemma *interaction-bound-map-gpv′-le*:
defines *ib* ≡ *interaction-bound*
shows *interaction-bound consider* (*map-gpv′ f g h gpv*) ≤ *ib* (*consider* ∘ *g*) *gpv*
⟨*proof*⟩

lemma *interaction-bounded-by-map-gpv′* [*interaction-bound*]:
assumes *interaction-bounded-by* (*consider* ∘ *g*) *gpv n*
shows *interaction-bounded-by consider* (*map-gpv′ f g h gpv*) *n*
⟨*proof*⟩

lemma *map-gpv′-bind-gpv*:
map-gpv′ f g h (*bind-gpv gpv F*) = *bind-gpv* (*map-gpv′ id g h gpv*) (λ*x*. *map-gpv′ f g h* (*F x*))
⟨*proof*⟩

lemma *exec-gpv-map-gpv′*:
exec-gpv callee (*map-gpv′ f g h gpv*) *s* =
map-spmf (*map-prod f id*) (*exec-gpv* (*map-fun id* (*map-fun g* (*map-spmf* (*map-prod h id*))) *callee*) *gpv s*)
⟨*proof*⟩

lemma *colossless-gpv-sub-gpvs*:
assumes *colossless-gpv ℐ gpv gpv′* ∈ *sub-gpvs ℐ gpv*
shows *colossless-gpv ℐ gpv′*
⟨*proof*⟩

lemma *pfinite-gpv-sub-gpvs*:
assumes *pfinite-gpv ℐ gpv gpv′* ∈ *sub-gpvs ℐ gpv ℐ* ⊢*g gpv* √
shows *pfinite-gpv ℐ gpv′*
⟨*proof*⟩

lemma *pfinite-gpv-id-oracle* [*simp*]: *pfinite-gpv ℐ* (*id-oracle s x*) **if** *x* ∈ *outs-ℐ ℐ*
⟨*proof*⟩

2.1 *try-gpv*

lemma *plossless-gpv-try-gpvI*:
assumes *pfinite-gpv ℐ gpv*
and ¬ *colossless-gpv ℐ gpv* ⟹ *plossless-gpv ℐ gpv′*
shows *plossless-gpv ℐ* (*TRY gpv ELSE gpv′*)
⟨*proof*⟩

lemma *WT-gpv-try-gpvI* [*WT-intro*]:
assumes $\mathcal{I} \vdash g\ gpv\ \surd$
and $\neg$ *colossless-gpv* $\mathcal{I}$ *gpv* $\Longrightarrow \mathcal{I} \vdash g\ gpv'\ \surd$
shows $\mathcal{I} \vdash g$ *try-gpv gpv gpv'* $\surd$
⟨*proof*⟩

lemma (**in** *callee-invariant-on*) *exec-gpv-try-gpv*:
fixes *exec-gpv1*
defines *exec-gpv1* $\equiv$ *exec-gpv*
assumes *WT*: $\mathcal{I} \vdash g\ gpv\ \surd$
and *pfinite*: *pfinite-gpv* $\mathcal{I}$ *gpv*
and *I*: *I s*
and *f*: $\bigwedge s.\ I\ s \Longrightarrow f\ (x,\ s) = z$
and *lossless*: $\bigwedge s\ x.\ [\![x \in$ *outs-*$\mathcal{I}$; *I s*$]\!] \Longrightarrow$ *lossless-spmf* (*callee s x*)
shows *map-spmf f* (*exec-gpv callee* (*try-gpv gpv* (*Done x*)) *s*) =
try-spmf (*map-spmf f* (*exec-gpv1 callee gpv s*)) (*return-spmf z*)
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *try-gpv-bind-gen-lossless'*: — generalises *gen-lossless-gpv ?b* $\mathcal{I}$*-full ?gpv* $\Longrightarrow$ *TRY ?gpv* $\gg\!=$ *?f ELSE ?gpv' = ?gpv* $\gg\!=$ ($\lambda x.$ *TRY ?f x ELSE ?gpv'*)
assumes *lossless*: *gen-lossless-gpv b* $\mathcal{I}$ *gpv*
and *WT1*: $\mathcal{I} \vdash g\ gpv\ \surd$
and *WT2*: $\mathcal{I} \vdash g\ gpv'\ \surd$
and *WTf*: $\bigwedge x.\ x \in$ *results-gpv* $\mathcal{I}$ *gpv* $\Longrightarrow \mathcal{I} \vdash g\ f\ x\ \surd$
shows *eq-*$\mathcal{I}$*-gpv* (=) $\mathcal{I}$ (*TRY bind-gpv gpv f ELSE gpv'*) (*bind-gpv gpv* ($\lambda x.$ *TRY f x ELSE gpv'*))
⟨*proof*⟩
lemmas *try-gpv-bind-lossless' = try-gpv-bind-gen-lossless'*[**where** *b=False*]
and *try-gpv-bind-colossless' = try-gpv-bind-gen-lossless'*[**where** *b=True*]

lemma *try-gpv-bind-gpv*:
try-gpv (*bind-gpv gpv f*) *gpv'* =
bind-gpv (*try-gpv* (*map-gpv Some id gpv*) (*Done None*)) ($\lambda x.$ *case x of None* $\Rightarrow$ *gpv'* | *Some x'* $\Rightarrow$ *try-gpv* (*f x'*) *gpv'*)
⟨*proof*⟩

lemma *bind-gpv-try-gpv-map-Some*:
bind-gpv (*try-gpv* (*map-gpv Some id gpv*) (*Done None*)) ($\lambda x.$ *case x of None* $\Rightarrow$ *Fail* | *Some y* $\Rightarrow$ *f y*) =
bind-gpv gpv f
⟨*proof*⟩

lemma *try-gpv-left-gpv*:
assumes $\mathcal{I} \vdash g\ gpv\ \surd$ **and** *WT2*: $\mathcal{I} \vdash g\ gpv'\ \surd$
shows *eq-*$\mathcal{I}$*-gpv* (=) ($\mathcal{I} \oplus_{\mathcal{I}} \mathcal{I}'$) (*try-gpv* (*left-gpv gpv*) (*left-gpv gpv'*)) (*left-gpv* (*try-gpv gpv gpv'*))
⟨*proof*⟩

lemma *try-gpv-right-gpv*:
assumes $\mathcal{I}' \vdash g$ *gpv* $\surd$ **and** *WT2*: $\mathcal{I}' \vdash g$ *gpv'* $\surd$
shows *eq-*$\mathcal{I}$*-gpv* (=) ($\mathcal{I} \oplus_{\mathcal{I}} \mathcal{I}'$) (*try-gpv* (*right-gpv gpv*) (*right-gpv gpv'*)) (*right-gpv* (*try-gpv gpv gpv'*))
⟨*proof*⟩

lemma *bind-try-Done-Fail*: *bind-gpv* (*TRY gpv ELSE Done x*) *f* = *bind-gpv gpv f* **if** *f x* = *Fail*
⟨*proof*⟩

lemma *inline-map-gpv'*:
inline callee (*map-gpv' f g h gpv*) *s* =
map-gpv (*apfst f*) *id* (*inline* (*map-fun id* (*map-fun g* (*map-gpv* (*apfst h*) *id*)) *callee*) *gpv s*)
⟨*proof*⟩

lemma *interaction-bound-try-gpv*:
fixes *consider* **defines** *ib* ≡ *interaction-bound consider*
shows *interaction-bound consider* (*try-gpv gpv gpv'*) ≤ *ib gpv* + *ib gpv'*
⟨*proof*⟩

lemma *interaction-bounded-by-try-gpv* [*interaction-bound*]:
interaction-bounded-by consider (*try-gpv gpv1 gpv2*) (*bound1* + *bound2*)
if *interaction-bounded-by consider gpv1 bound1 interaction-bounded-by consider gpv2 bound2*
⟨*proof*⟩

2.2 term *gpv-stop*

lemma *interaction-bounded-by-gpv-stop* [*interaction-bound*]:
assumes *interaction-bounded-by consider gpv n*
shows *interaction-bounded-by consider* (*gpv-stop gpv*) *n*
⟨*proof*⟩

context includes $\mathcal{I}$*.lifting* **begin**

lift-definition *stop-*$\mathcal{I}$ :: ('*a*, '*b*) $\mathcal{I}$ ⇒ ('*a*, '*b option*) $\mathcal{I}$ **is**
λ*resp x*. *if* (*resp x* = {}) *then* {} *else insert None* (*Some* ‘ *resp x*) ⟨*proof*⟩

lemma *outs-stop-*$\mathcal{I}$ [*simp*]: *outs-*$\mathcal{I}$ (*stop-*$\mathcal{I}$ $\mathcal{I}$) = *outs-*$\mathcal{I}$ $\mathcal{I}$
⟨*proof*⟩

lemma *responses-stop-*$\mathcal{I}$ [*simp*]:
responses-$\mathcal{I}$ (*stop-*$\mathcal{I}$ $\mathcal{I}$) *x* = (*if x* ∈ *outs-*$\mathcal{I}$ $\mathcal{I}$ *then insert None* (*Some* ‘ *responses-*$\mathcal{I}$ $\mathcal{I}$ *x*) *else* {})
⟨*proof*⟩

lemma *stop-$\mathcal{I}$-full* [*simp*]: *stop-$\mathcal{I}$ $\mathcal{I}$-full = $\mathcal{I}$-full*
⟨*proof*⟩

lemma *stop-$\mathcal{I}$-uniform* [*simp*]:
stop-$\mathcal{I}$ ($\mathcal{I}$-uniform A B) = (if B = {} then ⊥ else $\mathcal{I}$-uniform A (insert None (Some ‘ B)))
⟨*proof*⟩

lifting-update *$\mathcal{I}$.lifting*
lifting-forget *$\mathcal{I}$.lifting*

end

lemma *stop-$\mathcal{I}$-bot* [*simp*]: *stop-$\mathcal{I}$ ⊥ = ⊥*
⟨*proof*⟩

lemma *WT-gpv-stop* [*simp*, *WT-intro*]: *stop-$\mathcal{I}$ $\mathcal{I}$ ⊢g gpv-stop gpv √* **if** *$\mathcal{I}$ ⊢g gpv √*
⟨*proof*⟩

lemma *expectation-gpv-stop*:
fixes *fail* **and** *gpv* :: (*'a*, *'b*, *'c*) *gpv*
assumes *WT*: *$\mathcal{I}$ ⊢g gpv √*
and *fail*: *fail ≤ c*
shows *expectation-gpv fail (stop-$\mathcal{I}$ $\mathcal{I}$) (λ-. c) (gpv-stop gpv) = expectation-gpv fail $\mathcal{I}$ (λ-. c) gpv* (**is** *?lhs = ?rhs*)
⟨*proof*⟩

lemma *pgen-lossless-gpv-stop*:
fixes *fail* **and** *gpv* :: (*'a*, *'b*, *'c*) *gpv*
assumes *WT*: *$\mathcal{I}$ ⊢g gpv √*
and *fail*: *fail ≤ 1*
shows *pgen-lossless-gpv fail (stop-$\mathcal{I}$ $\mathcal{I}$) (gpv-stop gpv) = pgen-lossless-gpv fail $\mathcal{I}$ gpv*
⟨*proof*⟩

lemma *pfinite-gpv-stop* [*simp*]:
pfinite-gpv (stop-$\mathcal{I}$ $\mathcal{I}$) (gpv-stop gpv) ⟷ pfinite-gpv $\mathcal{I}$ gpv **if** *$\mathcal{I}$ ⊢g gpv √*
⟨*proof*⟩

lemma *plossless-gpv-stop* [*simp*]:
plossless-gpv (stop-$\mathcal{I}$ $\mathcal{I}$) (gpv-stop gpv) ⟷ plossless-gpv $\mathcal{I}$ gpv **if** *$\mathcal{I}$ ⊢g gpv √*
⟨*proof*⟩

lemma *results-gpv-stop-SomeD*: *Some x ∈ results-gpv (stop-$\mathcal{I}$ $\mathcal{I}$) (gpv-stop gpv) ⟹ x ∈ results-gpv $\mathcal{I}$ gpv*
⟨*proof*⟩

lemma *Some-in-results'-gpv-gpv-stopD*: *Some xy ∈ results'-gpv (gpv-stop gpv) ⟹ xy ∈ results'-gpv gpv*

⟨*proof*⟩

2.3 term *exception-ℐ*

datatype *'s exception* = *Fault* | *OK* (*ok*: *'s*)

lemma *inj-on-OK* [*simp*]: *inj-on OK A*
⟨*proof*⟩

function *join-exception* :: *'a exception* ⇒ *'b exception* ⇒ (*'a* × *'b*) *exception* **where**
join-exception Fault - = Fault
| *join-exception - Fault = Fault*
| *join-exception* (*OK a*) (*OK b*) = *OK* (*a*, *b*)
⟨*proof*⟩
termination ⟨*proof*⟩

primrec *merge-exception* :: *'a exception* + *'b exception* ⇒ (*'a* + *'b*) *exception* **where**
merge-exception (*Inl x*) = *map-exception Inl x*
| *merge-exception* (*Inr y*) = *map-exception Inr y*

fun *option-of-exception* :: *'a exception* ⇒ *'a option* **where**
option-of-exception Fault = *None*
| *option-of-exception* (*OK x*) = *Some x*

fun *exception-of-option* :: *'a option* ⇒ *'a exception* **where**
exception-of-option None = *Fault*
| *exception-of-option* (*Some x*) = *OK x*

lemma *option-of-exception-exception-of-option* [*simp*]: *option-of-exception* (*exception-of-option x*) = *x*
⟨*proof*⟩

lemma *exception-of-option-option-of-exception* [*simp*]: *exception-of-option* (*option-of-exception x*) = *x*
⟨*proof*⟩

lemma *case-exception-of-option* [*simp*]: *case-exception f g* (*exception-of-option x*) = *case-option f g x*
⟨*proof*⟩

lemma *case-option-of-exception* [*simp*]: *case-option f g* (*option-of-exception x*) = *case-exception f g x*
⟨*proof*⟩

lemma *surj-exception-of-option* [*simp*]: *surj exception-of-option*
⟨*proof*⟩

lemma *surj-option-of-exception* [*simp*]: *surj option-of-exception*
⟨*proof*⟩

lemma *case-map-exception* [*simp*]: *case-exception f g* (*map-exception h x*) = *case-exception f* (*g* ∘ *h*) *x*
⟨*proof*⟩

definition *exception-ℐ* :: ('*a*, '*b*) *ℐ* ⇒ ('*a*, '*b exception*) *ℐ* **where**
exception-ℐ ℐ = *map-ℐ id exception-of-option* (*stop-ℐ ℐ*)

lemma *outs-exception-ℐ* [*simp*]: *outs-ℐ* (*exception-ℐ ℐ*) = *outs-ℐ ℐ*
⟨*proof*⟩

lemma *responses-exception-ℐ* [*simp*]:
responses-ℐ (*exception-ℐ ℐ*) *x* = (*if x* ∈ *outs-ℐ ℐ then insert Fault* (*OK* ‘ *responses-ℐ ℐ x*) *else* {})
⟨*proof*⟩

lemma *map-ℐ-full* [*simp*]: *map-ℐ f g ℐ-full* = *ℐ-uniform UNIV* (*range g*)
⟨*proof*⟩

lemma *exception-ℐ-full* [*simp*]: *exception-ℐ ℐ-full* = *ℐ-full*
⟨*proof*⟩

lemma *exception-ℐ-uniform* [*simp*]:
exception-ℐ (*ℐ-uniform A B*) = (*if B* = {} *then* ⊥ *else ℐ-uniform A* (*insert Fault* (*OK* ‘ *B*)))
⟨*proof*⟩

lemma *option-of-exception-ℐ* [*simp*]: *map-ℐ id option-of-exception* (*exception-ℐ ℐ*) = *stop-ℐ ℐ*
⟨*proof*⟩

lemma *exception-of-option-ℐ* [*simp*]: *map-ℐ id exception-of-option* (*stop-ℐ ℐ*) = *exception-ℐ ℐ*
⟨*proof*⟩

2.4 inline

context *raw-converter-invariant* **begin**

context
fixes *gpv* :: ('*a*, '*call*, '*ret*) *gpv*
assumes *gpv*: *plossless-gpv ℐ gpv ℐ* ⊢*g gpv* √
begin

lemma *lossless-spmf-inline1*:
assumes *lossless*: ⋀*s x*. ⟦ *x* ∈ *outs-ℐ ℐ*; *I s* ⟧ ⟹ *lossless-spmf* (*the-gpv* (*callee s x*))

and I: $I\ s$
shows *lossless-spmf* (*inline1 callee gpv s*)
⟨*proof*⟩

end

end

lemma (**in** *raw-converter-invariant*) *inline1-try-gpv*:
defines $inline1' \equiv inline1$
assumes WT: $\mathcal{I} \vdash g\ gpv\ \surd$
and *pfinite*: *pfinite-gpv* $\mathcal{I}$ *gpv*
and f: $\bigwedge s.\ I\ s \Longrightarrow f\ (x,\ s) = z$
and *lossless*: $\bigwedge s\ x.\ [\![\ x \in outs\text{-}\mathcal{I}\ \mathcal{I};\ I\ s\ ]\!] \Longrightarrow colossless\text{-}gpv\ \mathcal{I}'\ (callee\ s\ x)$
and I: $I\ s$
shows *map-spmf* (*map-sum f id*) (*inline1 callee* (*try-gpv gpv* (*Done x*)) *s*) =
try-spmf (*map-spmf* (*map-sum f* (λ(*out*, *c*, *rpv*). (*out*, *c*, λ*input*. *try-gpv* (*rpv input*) (*Done x*)))) (*inline1′ callee gpv s*)) (*return-spmf* (*Inl z*))
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma (**in** *raw-converter-invariant*) *inline-try-gpv*:
assumes WT: $\mathcal{I} \vdash g\ gpv\ \surd$
and *pfinite*: *pfinite-gpv* $\mathcal{I}$ *gpv*
and f: $\bigwedge s.\ I\ s \Longrightarrow f\ (x,\ s) = z$
and *lossless*: $\bigwedge s\ x.\ [\![\ x \in outs\text{-}\mathcal{I}\ \mathcal{I};\ I\ s\ ]\!] \Longrightarrow colossless\text{-}gpv\ \mathcal{I}'\ (callee\ s\ x)$
and I: $I\ s$
shows *eq-I-gpv* (=) $\mathcal{I}'$ (*map-gpv f id* (*inline callee* (*try-gpv gpv* (*Done x*)) *s*)) (*try-gpv* (*map-gpv f id* (*inline callee gpv s*)) (*Done z*))
(**is** *eq-I-gpv - - ?lhs ?rhs*)
⟨*proof*⟩

definition *cr-prod2* :: $'a \Rightarrow ('b \Rightarrow 'c \Rightarrow bool) \Rightarrow 'b \Rightarrow 'a \times 'c \Rightarrow bool$ **where**
cr-prod2 x A = (λ*b* (*a*, *c*). *A b c* ∧ *x* = *a*)

lemma *cr-prod2-simps* [*simp*]: *cr-prod2 x A a* (*b*, *c*) ⟷ *A a c* ∧ *x* = *b*
⟨*proof*⟩

lemma *cr-prod2I*: *A a b* ⟹ *cr-prod2 x A a* (*x*, *b*) ⟨*proof*⟩

lemma *cr-prod2-Grp*: *cr-prod2 x* (*BNF-Def.Grp A f*) = *BNF-Def.Grp A* (λ*b*. (*x*, *f b*))
⟨*proof*⟩

lemma *extend-state-oracle-transfer′*: **includes** *lifting-syntax* **shows**
((*S* ===> *C* ===> *rel-spmf* (*rel-prod R S*)) ===> *cr-prod2 s S* ===> *C* ===> *rel-spmf* (*rel-prod R* (*cr-prod2 s S*))) (λ*oracle*. *oracle*) *extend-state-oracle*

⟨*proof*⟩

lemma *exec-gpv-extend-state-oracle*:
exec-gpv (*extend-state-oracle callee*) *gpv* (*s*, *s'*) =
map-spmf (λ(*x*, *s''*). (*x*, (*s*, *s''*))) (*exec-gpv callee gpv s'*)
⟨*proof*⟩

3 Material for Constructive Crypto

lemma *WT-resource-I-uniform-UNIV* [*simp*]: *I-uniform A UNIV* ⊢*res res* √
⟨*proof*⟩

lemma *WT-converter-of-callee-invar*:
assumes *WT*: ⋀*s q*. ⟦ *q* ∈ *outs-I I*; *I s* ⟧ ⟹ *I'* ⊢*g callee s q* √
and *res*: ⋀*s q r s'*. ⟦ (*r*, *s'*) ∈ *results-gpv I'* (*callee s q*); *q* ∈ *outs-I I*; *I s* ⟧
⟹ *r* ∈ *responses-I I q* ∧ *I s'*
and *I*: *I s*
shows *I*, *I'* ⊢*C converter-of-callee callee s* √
⟨*proof*⟩

lemma *eq-I-gpv-eq-OO*:
assumes *eq-I-gpv* (=) *I gpv gpv'* *eq-I-gpv A I gpv' gpv''*
shows *eq-I-gpv A I gpv gpv''*
⟨*proof*⟩

lemma *eq-I-gpv-eq-OO2*:
assumes *eq-I-gpv* (=) *I gpv'' gpv'* *eq-I-gpv A I gpv gpv'*
shows *eq-I-gpv A I gpv gpv''*
⟨*proof*⟩

lemma *eq-I-gpv-try-gpv-cong*:
assumes *eq-I-gpv A I gpv1 gpv1'*
and *eq-I-gpv A I gpv2 gpv2'*
shows *eq-I-gpv A I* (*try-gpv gpv1 gpv2*) (*try-gpv gpv1' gpv2'*)
⟨*proof*⟩

lemma *eq-I-gpv-map-gpv'*:
assumes *eq-I-gpv* (*BNF-Def.vimage2p f f' A*) (*map-I g h I*) *gpv1 gpv2*
shows *eq-I-gpv A I* (*map-gpv' f g h gpv1*) (*map-gpv' f' g h gpv2*)
⟨*proof*⟩

lemma *eq-I-converter-map-converter*:
assumes *map-I* (*inv-into UNIV f*) (*inv-into UNIV g*) *I*, *map-I f' g' I'* ⊢*C conv1* ∼ *conv2*
and *inj f surj g*
shows *I*, *I'* ⊢*C map-converter f g f' g' conv1* ∼ *map-converter f g f' g' conv2*
⟨*proof*⟩

lemma *resource-of-oracle-run-resource*: *resource-of-oracle run-resource res* = *res*
⟨*proof*⟩

lemma *connect-map-gpv'*:
connect (*map-gpv'* *f g h adv*) *res* = *map-spmf f* (*connect adv* (*map-resource g h res*))
⟨*proof*⟩

primcorec *fail-resource* :: ('*a*, '*b*) *resource* **where**
run-resource fail-resource = (λ-. *return-pmf None*)

lemma *WT-fail-resource* [*WT-intro*]: $\mathcal{I} \vdash_{res}$ *fail-resource* $\surd$
⟨*proof*⟩

context fixes *y* :: '*b* **begin**

primcorec *const-resource* :: ('*a*, '*b*) *resource* **where**
run-resource const-resource = (λ-. *map-spmf* (*map-prod id* (λ-. *const-resource*)) (*return-spmf* (*y*, ()))))

end

lemma *const-resource-sel* [*simp*]: *run-resource* (*const-resource y*) = (λ-. *return-spmf* (*y*, *const-resource y*))
⟨*proof*⟩

declare *const-resource.sel* [*simp del*]

lemma *lossless-const-resource* [*simp*]: *lossless-resource* $\mathcal{I}$ (*const-resource y*)
⟨*proof*⟩

lemma *WT-const-resource* [*simp*]:
$\mathcal{I} \vdash_{res}$ *const-resource y* $\surd \longleftrightarrow$ ($\forall$ *x*∈*outs-*$\mathcal{I}$ $\mathcal{I}$. *y* ∈ *responses-*$\mathcal{I}$ $\mathcal{I}$ *x*) (**is** *?lhs* $\longleftrightarrow$ *?rhs*)
⟨*proof*⟩

context fixes *y* :: '*b* **begin**

primcorec *const-converter* :: ('*a*, '*b*, '*c*, '*d*) *converter* **where**
run-converter const-converter = (λ-. *map-gpv* (*map-prod id* (λ-. *const-converter*)) *id* (*Done* (*y*, ()))))

end

lemma *const-converter-sel* [*simp*]: *run-converter* (*const-converter y*) = (λ-. *Done* (*y*, *const-converter y*))
⟨*proof*⟩

lemma *attach-const-converter* [*simp*]: *attach* (*const-converter y*) *res* = *const-resource y*
⟨*proof*⟩

declare *const-converter.sel* [*simp del*]

lemma *comp-const-converter* [*simp*]: *comp-converter* (*const-converter x*) *conv* = *const-converter x*
⟨*proof*⟩

lemma *interaction-bounded-const-converter* [*simp*, *interaction-bound*]:
interaction-any-bounded-converter (*const-converter Fault*) *bound*
⟨*proof*⟩

primcorec *merge-exception-converter* :: ('*a*, ('*b* + '*c*) *exception*, '*a*, '*b exception* + '*c exception*) *converter* **where**
run-converter merge-exception-converter =
(λ*x*. *map-gpv* (*map-prod id* (λ*conv*. *case conv of None* ⇒ *merge-exception-converter* | *Some conv'* ⇒ *conv'*)) *id* (
Pause x (λ*y*. *Done* (*case merge-exception y of Fault* ⇒ (*Fault*, *Some* (*const-converter Fault*))
| *OK y'* ⇒ (*OK y'*, *None*)))))

lemma *merge-exception-converter-sel* [*simp*]:
run-converter merge-exception-converter x =
Pause x (λ*y*. *Done* (*case merge-exception y of Fault* ⇒ (*Fault*, *const-converter Fault*) | *OK y'* ⇒ (*OK y'*, *merge-exception-converter*)))
⟨*proof*⟩

declare *merge-exception-converter.sel*[*simp del*]

lemma *plossless-const-converter*[*simp*]: *plossless-converter* $\mathcal{I}$ $\mathcal{I}'$ (*const-converter x*)
⟨*proof*⟩

lemma *plossless-merge-exception-converter* [*simp*]:
plossless-converter (*exception-*$\mathcal{I}$ ($\mathcal{I}$ $\oplus_{\mathcal{I}}$ $\mathcal{I}'$)) (*exception-*$\mathcal{I}$ $\mathcal{I}$ $\oplus_{\mathcal{I}}$ *exception-*$\mathcal{I}$ $\mathcal{I}'$) *merge-exception-converter*
⟨*proof*⟩

lemma *WT-const-converter* [*WT-intro*, *simp*]:
$\mathcal{I}$, $\mathcal{I}'$ $\vdash_C$ *const-converter x* $\surd$ **if** ∀ *q* ∈ *outs-*$\mathcal{I}$ $\mathcal{I}$. *x* ∈ *responses-*$\mathcal{I}$ $\mathcal{I}$ *q*
⟨*proof*⟩

lemma *WT-merge-exception-converter* [*WT-intro*, *simp*]:
exception-$\mathcal{I}$ ($\mathcal{I}1' \oplus_{\mathcal{I}} \mathcal{I}2'$), *exception-*$\mathcal{I}$ $\mathcal{I}1' \oplus_{\mathcal{I}}$ *exception-*$\mathcal{I}$ $\mathcal{I}2' \vdash_C$ *merge-exception-converter* $\surd$
⟨*proof*⟩

lemma *inline-left-gpv-merge-exception-converter*:
bind-gpv (*inline run-converter* (*map-gpv′ id id option-of-exception* (*gpv-stop* (*left-gpv gpv*))) *merge-exception-converter*) (λ(*x*, *conv′*). *case x of None* ⇒ *Fail* | *Some x′* ⇒ *Done* (*x*, *conv′*)) =
bind-gpv (*left-gpv* (*map-gpv′ id id option-of-exception* (*gpv-stop gpv*))) (λ*x*. *case x of None* ⇒ *Fail* | *Some x′* ⇒ *Done* (*x*, *merge-exception-converter*))
⟨*proof*⟩

lemma *inline-right-gpv-merge-exception-converter*:
bind-gpv (*inline run-converter* (*map-gpv′ id id option-of-exception* (*gpv-stop* (*right-gpv gpv*))) *merge-exception-converter*) (λ(*x*, *conv′*). *case x of None* ⇒ *Fail* | *Some x′* ⇒ *Done* (*x*, *conv′*)) =
bind-gpv (*right-gpv* (*map-gpv′ id id option-of-exception* (*gpv-stop gpv*))) (λ*x*. *case x of None* ⇒ *Fail* | *Some x′* ⇒ *Done* (*x*, *merge-exception-converter*))
⟨*proof*⟩

3.1 *Constructive-Cryptography.Wiring*

abbreviation (*input*)
id-wiring :: ('*a*, '*b*, '*a*, '*b*) *wiring* (1_w)
where
id-wiring ≡ (*id*, *id*)

definition
swap-lassocr$_w$ :: ('*a* + '*b* + '*c*, '*d* + '*e* + '*f*, '*b* + '*a* + '*c*, '*e* + '*d* + '*f*) *wiring*
where
swap-lassocr$_w$ ≡ *rassocl*$_w$ ∘$_w$ ((*swap*$_w$ |$_w$ 1_w) ∘$_w$ *lassocr*$_w$)

schematic-goal
wiring-swap-lassocr[*wiring-intro*]: *wiring ?I1 ?I2 swap-lassocr swap-lassocr*$_w$
⟨*proof*⟩

definition
parallel-wiring$_w$ :: (('*a* + '*b*) + ('*c* + '*d*), ('*e* + '*f*) + ('*g* + '*h*),
('*a* + '*c*) + ('*b* + '*d*), ('*e* + '*g*) + ('*f* + '*h*)) *wiring*
where
parallel-wiring$_w$ ≡ *lassocr*$_w$ ∘$_w$ ((1_w |$_w$ *swap-lassocr*$_w$) ∘$_w$ *rassocl*$_w$)

schematic-goal
wiring-parallel-wiring[*wiring-intro*]: *wiring ?I1 ?I2 parallel-wiring parallel-wiring*$_w$
⟨*proof*⟩

lemma *lassocr-inverse*: *rassocl*$_C$ ⊙ *lassocr*$_C$ = 1_C
⟨*proof*⟩

lemma *rassocl-inverse*: *lassocr*$_C$ ⊙ *rassocl*$_C$ = 1_C
⟨*proof*⟩

lemma *swap-sum-swap-sum* [*simp*]: *swap-sum* (*swap-sum x*) = *x*
⟨*proof*⟩

lemma *inj-on-lsumr* [*simp*]: *inj-on lsumr A*
⟨*proof*⟩

lemma *inj-on-rsuml* [*simp*]: *inj-on rsuml A*
⟨*proof*⟩

lemma *bij-lsumr* [*simp*]: *bij lsumr*
⟨*proof*⟩

lemma *bij-swap-sum* [*simp*]: *bij swap-sum*
⟨*proof*⟩

lemma *bij-rsuml* [*simp*]: *bij rsuml*
⟨*proof*⟩

lemma *bij-lassocr-swap-sum* [*simp*]: *bij lassocr-swap-sum*
⟨*proof*⟩

lemma *inj-lassocr-swap-sum* [*simp*]: *inj lassocr-swap-sum*
⟨*proof*⟩

lemma *inv-rsuml* [*simp*]: *inv-into UNIV rsuml* = *lsumr*
⟨*proof*⟩

lemma *inv-lsumr* [*simp*]: *inv-into UNIV lsumr* = *rsuml*
⟨*proof*⟩

lemma *lassocr-swap-sum-inverse* [*simp*]: *lassocr-swap-sum* (*lassocr-swap-sum x*) = *x*
⟨*proof*⟩

lemma *inv-lassocr-swap-sum* [*simp*]: *inv-into UNIV lassocr-swap-sum* = *lassocr-swap-sum*
⟨*proof*⟩

lemma *swap-inverse*: $swap_C \odot swap_C = 1_C$
⟨*proof*⟩

lemma *swap-lassocr-inverse*: $\mathcal{I}1 \oplus_\mathcal{I} (\mathcal{I}2 \oplus_\mathcal{I} \mathcal{I}3), \mathcal{I}1 \oplus_\mathcal{I} (\mathcal{I}2 \oplus_\mathcal{I} \mathcal{I}3) \vdash_C$ *swap-lassocr* ⊙ *swap-lassocr* ∼ 1_C
(**is** $?\mathcal{I}$,- ⊢$_C$ *?lhs* ∼ -)
⟨*proof*⟩

lemma *parallel-wiring-inverse*:
$(\mathcal{I}1 \oplus_\mathcal{I} \mathcal{I}2) \oplus_\mathcal{I} (\mathcal{I}3 \oplus_\mathcal{I} \mathcal{I}4), (\mathcal{I}1 \oplus_\mathcal{I} \mathcal{I}2) \oplus_\mathcal{I} (\mathcal{I}3 \oplus_\mathcal{I} \mathcal{I}4) \vdash_C$ *parallel-wiring* ⊙ *parallel-wiring* ∼ 1_C

(**is** *?I*, - $\vdash_C$ *?lhs* $\sim$ -)
⟨*proof*⟩
definition
attach-wiring-right ::
('*a*, '*b*, '*c*, '*d*) *wiring* ⇒
('*s* ⇒ '*e* ⇒ ('*f* × '*s*, '*a*, '*b*) *gpv*) ⇒ ('*s* ⇒ '*e* ⇒ ('*f* × '*s*, '*c*, '*d*) *gpv*)
where
attach-wiring-right = (λ(*f*, *g*). *map-fun id* (*map-fun id* (*map-gpv'* *id f g*)))

lemma
attach-wiring-right-simps:
attach-wiring-right (*f*, *g*) = *map-fun id* (*map-fun id* (*map-gpv'* *id f g*))
⟨*proof*⟩

lemma
comp-converter-of-callee-wiring:
assumes *wiring*: *wiring* $\mathcal{I}2$ $\mathcal{I}3$ *conv w*
and *WT*: $\mathcal{I}1$, $\mathcal{I}2$ $\vdash_C$ *CNV callee s* $\surd$
shows $\mathcal{I}1$, $\mathcal{I}3$ $\vdash_C$ *CNV callee s* $\odot$ *conv* $\sim$ *CNV* (*attach-wiring-right w callee*) *s*
⟨*proof*⟩

lemma *attach-wiring-right-comp-wiring*:
attach-wiring-right (*w1* $\circ_w$ *w2*) *callee* = *attach-wiring-right w2* (*attach-wiring-right w1 callee*)
⟨*proof*⟩

lemma *attach-wiring-comp-wiring*:
attach-wiring (*w1* $\circ_w$ *w2*) *callee* = *attach-wiring w1* (*attach-wiring w2 callee*)
⟨*proof*⟩

3.2 Probabilistic finite converter

coinductive *pfinite-converter* :: ('*a*, '*b*) $\mathcal{I}$ ⇒ ('*c*, '*d*) $\mathcal{I}$ ⇒ ('*a*, '*b*, '*c*, '*d*) *converter* ⇒ *bool*
for $\mathcal{I}$ $\mathcal{I}'$ **where**
pfinite-converterI: *pfinite-converter* $\mathcal{I}$ $\mathcal{I}'$ *conv* **if**
$\bigwedge a$. *a* ∈ *outs-*$\mathcal{I}$ $\mathcal{I}$ ⟹ *pfinite-gpv* $\mathcal{I}'$ (*run-converter conv a*)
$\bigwedge a$ *b conv'*. ⟦ *a* ∈ *outs-*$\mathcal{I}$ $\mathcal{I}$; (*b*, *conv'*) ∈ *results-gpv* $\mathcal{I}'$ (*run-converter conv a*) ⟧ ⟹ *pfinite-converter* $\mathcal{I}$ $\mathcal{I}'$ *conv'*

lemma *pfinite-converter-coinduct*[*consumes 1*, *case-names pfinite-converter*, *case-conclusion pfinite-converter pfinite step*, *coinduct pred*: *pfinite-converter*]:
assumes *X conv*
and *step*: $\bigwedge$*conv a*. ⟦ *X conv*; *a* ∈ *outs-*$\mathcal{I}$ $\mathcal{I}$ ⟧ ⟹ *pfinite-gpv* $\mathcal{I}'$ (*run-converter conv a*) ∧
(∀(*b*, *conv'*) ∈ *results-gpv* $\mathcal{I}'$ (*run-converter conv a*). *X conv'* ∨ *pfinite-converter* $\mathcal{I}$ $\mathcal{I}'$ *conv'*)
shows *pfinite-converter* $\mathcal{I}$ $\mathcal{I}'$ *conv*
⟨*proof*⟩

lemma *pfinite-converterD*:
⟦ *pfinite-converter* $\mathcal{I}$ $\mathcal{I}'$ *conv*; $a \in$ *outs-*$\mathcal{I}$ $\mathcal{I}$ ⟧
$\Longrightarrow$ *pfinite-gpv* $\mathcal{I}'$ (*run-converter conv a*) $\wedge$
$(\forall (b,\ conv') \in$ *results-gpv* $\mathcal{I}'$ (*run-converter conv a*). *pfinite-converter* $\mathcal{I}$ $\mathcal{I}'$ *conv'*)
⟨*proof*⟩

lemma *pfinite-converter-bot1* [*simp*]: *pfinite-converter bot* $\mathcal{I}$ *conv*
⟨*proof*⟩

lemma *pfinite-converter-mono*:
assumes $*$: *pfinite-converter* $\mathcal{I}1$ $\mathcal{I}2$ *conv*
and *le*: *outs-*$\mathcal{I}$ $\mathcal{I}1' \subseteq$ *outs-*$\mathcal{I}$ $\mathcal{I}1$ $\mathcal{I}2 \leq \mathcal{I}2'$
and *WT*: $\mathcal{I}1, \mathcal{I}2 \vdash_C$ *conv* $\surd$
shows *pfinite-converter* $\mathcal{I}1'$ $\mathcal{I}2'$ *conv*
⟨*proof*⟩

context *raw-converter-invariant* **begin**
lemma *pfinite-converter-of-callee*:
assumes *step*: $\bigwedge x\ s.$ ⟦ $x \in$ *outs-*$\mathcal{I}$ $\mathcal{I}$; *I s* ⟧ $\Longrightarrow$ *pfinite-gpv* $\mathcal{I}'$ (*callee s x*)
and *I*: *I s*
shows *pfinite-converter* $\mathcal{I}$ $\mathcal{I}'$ (*converter-of-callee callee s*)
⟨*proof*⟩
end

lemma *raw-converter-invariant-run-pfinite-converter*:
raw-converter-invariant $\mathcal{I}$ $\mathcal{I}'$ *run-converter* (λ*conv*. *pfinite-converter* $\mathcal{I}$ $\mathcal{I}'$ *conv* $\wedge$ $\mathcal{I}$,$\mathcal{I}' \vdash_C$ *conv* $\surd$)
⟨*proof*⟩

interpretation *run-pfinite-converter*: *raw-converter-invariant*
$\mathcal{I}$ $\mathcal{I}'$ *run-converter* λ*conv*. *pfinite-converter* $\mathcal{I}$ $\mathcal{I}'$ *conv* $\wedge$ $\mathcal{I}$,$\mathcal{I}' \vdash_C$ *conv* $\surd$ **for** $\mathcal{I}$ $\mathcal{I}'$
⟨*proof*⟩

named-theorems *pfinite-intro Introduction rules for probabilistic finiteness*

lemma *pfinite-id-converter* [*pfinite-intro*]: *pfinite-converter* $\mathcal{I}$ $\mathcal{I}$ *id-converter*
⟨*proof*⟩

lemma *pfinite-fail-converter* [*pfinite-intro*]: *pfinite-converter* $\mathcal{I}$ $\mathcal{I}'$ *fail-converter*
⟨*proof*⟩

lemma *pfinite-parallel-converter2* [*pfinite-intro*]:
pfinite-converter ($\mathcal{I}1 \oplus_{\mathcal{I}} \mathcal{I}2$) ($\mathcal{I}1' \oplus_{\mathcal{I}} \mathcal{I}2'$) (*conv1* $|_=$ *conv2*)
if *pfinite-converter* $\mathcal{I}1$ $\mathcal{I}1'$ *conv1 pfinite-converter* $\mathcal{I}2$ $\mathcal{I}2'$ *conv2*
⟨*proof*⟩

context *raw-converter-invariant* **begin**

lemma *expectation-gpv-1-le-inline*:
defines *expectation-gpv2* ≡ *expectation-gpv 1* $\mathcal{I}'$
assumes *callee*: $\bigwedge s\ x.\ [\![\ x \in \textit{outs-}\mathcal{I}\ \mathcal{I};\ I\ s\ ]\!] \Longrightarrow \textit{pfinite-gpv}\ \mathcal{I}'\ (\textit{callee}\ s\ x)$
and *WT-gpv*: $\mathcal{I} \vdash g\ \textit{gpv}\ \surd$
and *I*: *I s*
and *f-le-1*: $\bigwedge x.\ f\ x \leq 1$
shows *expectation-gpv 1* $\mathcal{I}$ *f gpv* ≤ *expectation-gpv2* (λ(*x*, *s*). *f x*) (*inline callee gpv s*)
⟨*proof*⟩

lemma *pfinite-inline*:
assumes *fin*: *pfinite-gpv* $\mathcal{I}$ *gpv*
and *WT*: $\mathcal{I} \vdash g\ \textit{gpv}\ \surd$
and *callee*: $\bigwedge s\ x.\ [\![\ x \in \textit{outs-}\mathcal{I}\ \mathcal{I};\ I\ s\ ]\!] \Longrightarrow \textit{pfinite-gpv}\ \mathcal{I}'\ (\textit{callee}\ s\ x)$
and *I*: *I s*
shows *pfinite-gpv* $\mathcal{I}'$ (*inline callee gpv s*)
⟨*proof*⟩

end

lemma *pfinite-comp-converter* [*pfinite-intro*]:
pfinite-converter $\mathcal{I}1$ $\mathcal{I}3$ (*conv1* ⊙ *conv2*)
if *pfinite-converter* $\mathcal{I}1$ $\mathcal{I}2$ *conv1 pfinite-converter* $\mathcal{I}2$ $\mathcal{I}3$ *conv2* $\mathcal{I}1$,$\mathcal{I}2$ $\vdash_C$ *conv1* $\surd$ $\mathcal{I}2$,$\mathcal{I}3$ $\vdash_C$ *conv2* $\surd$
⟨*proof*⟩

lemma *pfinite-map-converter* [*pfinite-intro*]:
pfinite-converter $\mathcal{I}$ $\mathcal{I}'$ (*map-converter f g f' g' conv*) **if**
∗: *pfinite-converter* (*map-*$\mathcal{I}$ (*inv-into UNIV f*) (*inv-into UNIV g*) $\mathcal{I}$) (*map-*$\mathcal{I}$ *f' g'* $\mathcal{I}'$) *conv*
and *f*: *inj f* **and** *g*: *surj g*
⟨*proof*⟩

lemma *pfinite-lassocr*$_C$ [*pfinite-intro*]: *pfinite-converter* (($\mathcal{I}1$ ⊕$_\mathcal{I}$ $\mathcal{I}2$) ⊕$_\mathcal{I}$ $\mathcal{I}3$) ($\mathcal{I}1$ ⊕$_\mathcal{I}$ ($\mathcal{I}2$ ⊕$_\mathcal{I}$ $\mathcal{I}3$)) *lassocr*$_C$
⟨*proof*⟩

lemma *pfinite-rassocl*$_C$ [*pfinite-intro*]: *pfinite-converter* ($\mathcal{I}1$ ⊕$_\mathcal{I}$ ($\mathcal{I}2$ ⊕$_\mathcal{I}$ $\mathcal{I}3$)) (($\mathcal{I}1$ ⊕$_\mathcal{I}$ $\mathcal{I}2$) ⊕$_\mathcal{I}$ $\mathcal{I}3$) *rassocl*$_C$
⟨*proof*⟩

lemma *pfinite-swap*$_C$ [*pfinite-intro*]: *pfinite-converter* ($\mathcal{I}1$ ⊕$_\mathcal{I}$ $\mathcal{I}2$) ($\mathcal{I}2$ ⊕$_\mathcal{I}$ $\mathcal{I}1$) *swap*$_C$
⟨*proof*⟩

lemma *pfinite-swap-lassocr* [*pfinite-intro*]: *pfinite-converter* ($\mathcal{I}1$ ⊕$_\mathcal{I}$ ($\mathcal{I}2$ ⊕$_\mathcal{I}$ $\mathcal{I}3$)) ($\mathcal{I}2$ ⊕$_\mathcal{I}$ ($\mathcal{I}1$ ⊕$_\mathcal{I}$ $\mathcal{I}3$)) *swap-lassocr*
⟨*proof*⟩

lemma *pfinite-swap-rassocl* [*pfinite-intro*]: *pfinite-converter* ((*I1* $\oplus_{\mathcal{I}}$ *I2*) $\oplus_{\mathcal{I}}$ *I3*) ((*I1* $\oplus_{\mathcal{I}}$ *I3*) $\oplus_{\mathcal{I}}$ *I2*) *swap-rassocl*
⟨*proof*⟩

lemma *pfinite-parallel-wiring* [*pfinite-intro*]:
pfinite-converter ((*I1* $\oplus_{\mathcal{I}}$ *I2*) $\oplus_{\mathcal{I}}$ (*I3* $\oplus_{\mathcal{I}}$ *I4*)) ((*I1* $\oplus_{\mathcal{I}}$ *I3*) $\oplus_{\mathcal{I}}$ (*I2* $\oplus_{\mathcal{I}}$ *I4*)) *parallel-wiring*
⟨*proof*⟩

lemma *pfinite-parallel-converter* [*pfinite-intro*]:
pfinite-converter (*I1* $\oplus_{\mathcal{I}}$ *I2*) *I3* (*conv1* $|_{\propto}$ *conv2*)
if *pfinite-converter I1 I3 conv1* **and** *pfinite-converter I2 I3 conv2*
⟨*proof*⟩

lemma *pfinite-converter-of-resource* [*simp*, *pfinite-intro*]: *pfinite-converter I1 I2* (*converter-of-resource res*)
⟨*proof*⟩

3.3 colossless converter

coinductive *colossless-converter* :: ('*a*, '*b*) $\mathcal{I}$ ⇒ ('*c*, '*d*) $\mathcal{I}$ ⇒ ('*a*, '*b*, '*c*, '*d*) *converter* ⇒ *bool*
for $\mathcal{I}$ $\mathcal{I}'$ **where**
colossless-converterI:
colossless-converter $\mathcal{I}$ $\mathcal{I}'$ *conv* **if**
⋀*a*. *a* ∈ *outs-I* $\mathcal{I}$ ⟹ *colossless-gpv* $\mathcal{I}'$ (*run-converter conv a*)
⋀*a b conv'*. ⟦ *a* ∈ *outs-I* $\mathcal{I}$; (*b*, *conv'*) ∈ *results-gpv* $\mathcal{I}'$ (*run-converter conv a*) ⟧ ⟹ *colossless-converter* $\mathcal{I}$ $\mathcal{I}'$ *conv'*

lemma *colossless-converter-coinduct*[*consumes 1*, *case-names colossless-converter*, *case-conclusion colossless-converter plossless step*, *coinduct pred*: *colossless-converter*]:
assumes *X conv*
and *step*: ⋀*conv a*. ⟦ *X conv*; *a* ∈ *outs-I* $\mathcal{I}$ ⟧ ⟹ *colossless-gpv* $\mathcal{I}'$ (*run-converter conv a*) ∧
(∀(*b*, *conv'*) ∈ *results-gpv* $\mathcal{I}'$ (*run-converter conv a*). *X conv'* ∨ *colossless-converter* $\mathcal{I}$ $\mathcal{I}'$ *conv'*)
shows *colossless-converter* $\mathcal{I}$ $\mathcal{I}'$ *conv*
⟨*proof*⟩

lemma *colossless-converterD*:
⟦ *colossless-converter* $\mathcal{I}$ $\mathcal{I}'$ *conv*; *a* ∈ *outs-I* $\mathcal{I}$ ⟧
⟹ *colossless-gpv* $\mathcal{I}'$ (*run-converter conv a*) ∧
(∀(*b*, *conv'*) ∈ *results-gpv* $\mathcal{I}'$ (*run-converter conv a*). *colossless-converter* $\mathcal{I}$ $\mathcal{I}'$ *conv'*)
⟨*proof*⟩

lemma *colossless-converter-bot1* [*simp*]: *colossless-converter bot* $\mathcal{I}$ *conv*
⟨*proof*⟩

lemma *raw-converter-invariant-run-colossless-converter*: *raw-converter-invariant* $\mathcal{I}$ $\mathcal{I}'$ *run-converter* (λ*conv. colossless-converter* $\mathcal{I}$ $\mathcal{I}'$ *conv* $\wedge$ $\mathcal{I}$,$\mathcal{I}'$ $\vdash_C$ *conv* $\surd$)
⟨*proof*⟩

interpretation *run-colossless-converter*: *raw-converter-invariant*
$\mathcal{I}$ $\mathcal{I}'$ *run-converter* λ*conv. colossless-converter* $\mathcal{I}$ $\mathcal{I}'$ *conv* $\wedge$ $\mathcal{I}$,$\mathcal{I}'$ $\vdash_C$ *conv* $\surd$ **for** $\mathcal{I}$ $\mathcal{I}'$
⟨*proof*⟩

lemma *colossless-const-converter* [*simp*]: *colossless-converter* $\mathcal{I}$ $\mathcal{I}'$ (*const-converter x*)
⟨*proof*⟩

3.4 trace equivalence

lemma *distinguish-trace-eq*:
assumes *distinguish*: $\bigwedge$*distinguisher*. $\mathcal{I}$ $\vdash g$ *distinguisher* $\surd$ $\Longrightarrow$ *connect distinguisher res* = *connect distinguisher res'*
shows *outs-*$\mathcal{I}$ $\mathcal{I}$ $\vdash_R$ *res* $\approx$ *res'*
⟨*proof*⟩

lemma *attach-trace-eq'*:
assumes *eq*: *outs-*$\mathcal{I}$ $\mathcal{I}$ $\vdash_R$ *res1* $\approx$ *res2*
and *WT1* [*WT-intro*]: $\mathcal{I}$ $\vdash$*res res1* $\surd$
and *WT2* [*WT-intro*]: $\mathcal{I}$ $\vdash$*res res2* $\surd$
and *WT-conv* [*WT-intro*]: $\mathcal{I}'$,$\mathcal{I}$ $\vdash_C$ *conv* $\surd$
shows *outs-*$\mathcal{I}$ $\mathcal{I}'$ $\vdash_R$ *conv* $\triangleright$ *res1* $\approx$ *conv* $\triangleright$ *res2*
⟨*proof*⟩

lemma *trace-callee-eq-trans* [*trans*]:
⟦ *trace-callee-eq callee1 callee2 A p q*; *trace-callee-eq callee2 callee3 A q r* ⟧
$\Longrightarrow$ *trace-callee-eq callee1 callee3 A p r*
⟨*proof*⟩

lemma *trace-eq'-parallel-resource*:
fixes *res1* :: ('*a*, '*b*) *resource* **and** *res2* :: ('*c*, '*d*) *resource*
assumes *1*: *trace-eq' A res1 res1'*
and *2*: *trace-eq' B res2 res2'*
shows *trace-eq'* (*A* <+> *B*) (*res1* ∥ *res2*) (*res1'* ∥ *res2'*)
⟨*proof*⟩

proposition *trace-callee-eq-coinduct* [*consumes 1*, *case-names step sim*]:
fixes *callee1* :: ('*a*, '*b*, '*s1*) *callee* **and** *callee2* :: ('*a*, '*b*, '*s2*) *callee*
assumes *start*: *S p q*
and *step*: $\bigwedge$*p q a*. ⟦ *S p q*; *a* ∈ *A* ⟧ $\Longrightarrow$
bind-spmf p (λ*s. map-spmf fst* (*callee1 s a*)) = *bind-spmf q* (λ*s. map-spmf fst* (*callee2 s a*))
and *sim*: $\bigwedge$*p q a res res' b s'' s'*. ⟦ *S p q*; *a* ∈ *A*; *res* ∈ *set-spmf p*; (*b*, *s''*) ∈

set-spmf (*callee1 res a*); *res'* ∈ *set-spmf q*; (*b*, *s'*) ∈ *set-spmf* (*callee2 res' a*) ⟧
⟹ *S* (*cond-spmf-fst* (*bind-spmf p* (λ*s*. *callee1 s a*)) *b*)
(*cond-spmf-fst* (*bind-spmf q* (λ*s*. *callee2 s a*)) *b*)
shows *trace-callee-eq callee1 callee2 A p q*
⟨*proof*⟩

proposition *trace-callee-eq-coinduct-strong* [*consumes 1*, *case-names step sim*, *case-conclusion step lhs rhs*, *case-conclusion sim sim eq*]:
fixes *callee1* :: ('*a*, '*b*, '*s1*) *callee* **and** *callee2* :: ('*a*, '*b*, '*s2*) *callee*
assumes *start*: *S p q*
and *step*: ⋀*p q a*. ⟦ *S p q*; *a* ∈ *A* ⟧ ⟹
bind-spmf p (λ*s*. *map-spmf fst* (*callee1 s a*)) = *bind-spmf q* (λ*s*. *map-spmf fst* (*callee2 s a*))
and *sim*: ⋀*p q a res res' b s'' s'*. ⟦ *S p q*; *a* ∈ *A*; *res* ∈ *set-spmf p*; (*b*, *s''*) ∈ *set-spmf* (*callee1 res a*); *res'* ∈ *set-spmf q*; (*b*, *s'*) ∈ *set-spmf* (*callee2 res' a*) ⟧
⟹ *S* (*cond-spmf-fst* (*bind-spmf p* (λ*s*. *callee1 s a*)) *b*)
(*cond-spmf-fst* (*bind-spmf q* (λ*s*. *callee2 s a*)) *b*) ∨
trace-callee-eq callee1 callee2 A (*cond-spmf-fst* (*bind-spmf p* (λ*s*. *callee1 s a*)) *b*) (*cond-spmf-fst* (*bind-spmf q* (λ*s*. *callee2 s a*)) *b*)
shows *trace-callee-eq callee1 callee2 A p q*
⟨*proof*⟩

lemma *trace-callee-return-pmf-None* [*simp*]:
trace-callee-eq callee1 callee2 A (*return-pmf None*) (*return-pmf None*)
⟨*proof*⟩

lemma *trace-callee-eq-sym* [*sym*]: *trace-callee-eq callee1 callee2 A p q* ⟹ *trace-callee-eq callee2 callee1 A q p*
⟨*proof*⟩

lemma *eq-resource-on-imp-trace-eq*: *A* ⊢$_R$ *res1* ≈ *res2* **if** *A* ⊢$_R$ *res1* ∼ *res2*
⟨*proof*⟩

lemma *advantage-nonneg*: *0* ≤ *advantage* 𝒜 *res1 res2*
⟨*proof*⟩

lemma *comp-converter-of-resource-conv-parallel-converter*:
(*converter-of-resource res* |$_\propto$ *1*$_C$) ⊙ *conv* = *converter-of-resource res* |$_\propto$ *conv*
⟨*proof*⟩

lemma *comp-converter-of-resource-conv-parallel-converter2*:
(*1*$_C$ |$_\propto$ *converter-of-resource res*) ⊙ *conv* = *conv* |$_\propto$ *converter-of-resource res*
⟨*proof*⟩

lemma *parallel-converter-map-converter*:
map-converter f g f' g' conv1 |$_\propto$ *map-converter f'' g'' f' g' conv2* =
map-converter (*map-sum f f''*) (*map-sum g g''*) *f' g'* (*conv1* |$_\propto$ *conv2*)
⟨*proof*⟩

lemma *map-converter-parallel-converter-out2*:
conv1 $|_\propto$ *map-converter f g id id conv2* = *map-converter* (*map-sum id f*) (*map-sum id g*) *id id* (*conv1* $|_\propto$ *conv2*)
⟨*proof*⟩

lemma *parallel-converter-assoc2*:
parallel-converter conv1 (*parallel-converter conv2 conv3*) =
map-converter lsumr rsuml id id (*parallel-converter* (*parallel-converter conv1 conv2*) *conv3*)
⟨*proof*⟩

lemma *parallel-converter-of-resource*:
converter-of-resource res1 $|_\propto$ *converter-of-resource res2* = *converter-of-resource* (*res1* ‖ *res2*)
⟨*proof*⟩

lemma *map-Inr-parallel-converter*:
map-converter Inr f g h (*conv1* $|_\propto$ *conv2*) = *map-converter id* (*f* ∘ *Inr*) *g h conv2*
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *map-Inl-parallel-converter*:
map-converter Inl f g h (*conv1* $|_\propto$ *conv2*) = *map-converter id* (*f* ∘ *Inl*) *g h conv1*
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *left-interface-parallel-converter*:
left-interface (*conv1* $|_\propto$ *conv2*) = *left-interface conv1* $|_\propto$ *left-interface conv2*
⟨*proof*⟩

lemma *right-interface-parallel-converter*:
right-interface (*conv1* $|_\propto$ *conv2*) = *right-interface conv1* $|_\propto$ *right-interface conv2*
⟨*proof*⟩

lemma *left-interface-converter-of-resource* [*simp*]:
left-interface (*converter-of-resource res*) = *converter-of-resource res*
⟨*proof*⟩

lemma *right-interface-converter-of-resource* [*simp*]:
right-interface (*converter-of-resource res*) = *converter-of-resource res*
⟨*proof*⟩

lemma *parallel-converter-swap*: *map-converter swap-sum swap-sum id id* (*conv1* $|_\propto$ *conv2*) = *conv2* $|_\propto$ *conv1*
⟨*proof*⟩

lemma *eq-𝓘-converter-map-converter′*:
assumes $\mathcal{I}''$, *map-𝓘 f′ g′* $\mathcal{I}'$ $\vdash_C$ *conv1* ∼ *conv2*
and *f* ‘ *outs-𝓘* $\mathcal{I}$ $\mathcal{I}$ ⊆ *outs-𝓘* $\mathcal{I}$ $\mathcal{I}''$

and $\forall q \in outs\text{-}\mathcal{I}\ \mathcal{I}.\ g$ ‘ $responses\text{-}\mathcal{I}\ \mathcal{I}''\ (f\ q) \subseteq responses\text{-}\mathcal{I}\ \mathcal{I}\ q$
shows $\mathcal{I}, \mathcal{I}' \vdash_C map\text{-}converter\ f\ g\ f'\ g'\ conv1 \sim map\text{-}converter\ f\ g\ f'\ g'\ conv2$
⟨*proof*⟩

lemma *parallel-converter-eq-I-cong*:
$[\![\mathcal{I}1, \mathcal{I} \vdash_C conv1 \sim conv1'; \mathcal{I}2, \mathcal{I} \vdash_C conv2 \sim conv2']\!]$
$\Longrightarrow \mathcal{I}1 \oplus_{\mathcal{I}} \mathcal{I}2, \mathcal{I} \vdash_C parallel\text{-}converter\ conv1\ conv2 \sim parallel\text{-}converter\ conv1'\ conv2'$
⟨*proof*⟩
lemma
exec-gpv-parallel-oracle-right:
exec-gpv (*oracle1* $\ddagger_O$ *oracle2*) (*right-gpv gpv*) *s* = *exec-gpv* (†*oracle2*) *gpv s*
⟨*proof*⟩

lemma
exec-gpv-parallel-oracle-left:
exec-gpv (*oracle1* $\ddagger_O$ *oracle2*) (*left-gpv gpv*) *s* = *exec-gpv* (*oracle1*†) *gpv s* (**is** *?L* = *?R*)
⟨*proof*⟩

end
theory *Observe-Failure* **imports**
More-CC
begin

declare [[*show-variants*]]

context fixes *oracle* :: (*'s*, *'in*, *'out*) *oracle'* **begin**

fun *obsf-oracle* :: (*'s exception*, *'in*, *'out exception*) *oracle'* **where**
obsf-oracle Fault x = *return-spmf* (*Fault*, *Fault*)
| *obsf-oracle* (*OK s*) *x* = *TRY map-spmf* (*map-prod OK OK*) (*oracle s x*) *ELSE return-spmf* (*Fault*, *Fault*)

end

type-synonym (*'a*, *'b*) *resource-obsf* = (*'a*, *'b exception*) *resource*

translations
(*type*) (*'a*, *'b*) *resource-obsf* <= (*type*) (*'a*, *'b exception*) *resource*

primcorec *obsf-resource* :: (*'in*, *'out*) *resource* ⇒ (*'in*, *'out*) *resource-obsf* **where**
run-resource (*obsf-resource res*) = (λ*x*.
map-spmf (*map-prod id obsf-resource*)
(*map-spmf* (*map-prod id* (λ*resF*. *case resF of OK res'* ⇒ *res'* | *Fault* ⇒ *fail-resource*))
(*TRY map-spmf* (*map-prod OK OK*) (*run-resource res x*) *ELSE return-spmf* (*Fault*, *Fault*))))

lemma *obsf-resource-sel*:
run-resource (*obsf-resource res*) *x* =
map-spmf (*map-prod id* (λ*resF*. *obsf-resource* (*case resF of OK res'* ⇒ *res'* | *Fault* ⇒ *fail-resource*)))
(*TRY map-spmf* (*map-prod OK OK*) (*run-resource res x*) *ELSE return-spmf* (*Fault*, *Fault*))
⟨*proof*⟩

declare *obsf-resource.simps* [*simp del*]

lemma *obsf-resource-exception* [*simp*]: *obsf-resource fail-resource* = *const-resource Fault*
⟨*proof*⟩

lemma *obsf-resource-sel2* [*simp*]:
run-resource (*obsf-resource res*) *x* =
try-spmf (*map-spmf* (*map-prod OK obsf-resource*) (*run-resource res x*)) (*return-spmf* (*Fault*, *const-resource Fault*))
⟨*proof*⟩

lemma *lossless-obsf-resource* [*simp*]: *lossless-resource* $\mathcal{I}$ (*obsf-resource res*)
⟨*proof*⟩

lemma *WT-obsf-resource* [*WT-intro*, *simp*]: *exception-*$\mathcal{I}$ $\mathcal{I}$ ⊢*res obsf-resource res* √ **if** $\mathcal{I}$ ⊢*res res* √
⟨*proof*⟩

type-synonym ('*a*, '*b*) *distinguisher-obsf* = (*bool*, '*a*, '*b exception*) *gpv*

translations
(*type*) ('*a*, '*b*) *distinguisher-obsf* <= (*type*) (*bool*, '*a*, '*b exception*) *gpv*

abbreviation *connect-obsf* :: ('*a*, '*b*) *distinguisher-obsf* ⇒ ('*a*, '*b*) *resource-obsf* ⇒ *bool spmf* **where**
connect-obsf == *connect*

definition *obsf-distinguisher* :: ('*a*, '*b*) *distinguisher* ⇒ ('*a*, '*b*) *distinguisher-obsf* **where**
obsf-distinguisher $\mathcal{D}$ = *map-gpv'* (λ*x*. *x* = *Some True*) *id option-of-exception* (*gpv-stop* $\mathcal{D}$)

lemma *WT-obsf-distinguisher* [*WT-intro*]:
exception-$\mathcal{I}$ $\mathcal{I}$ ⊢*g obsf-distinguisher* $\mathcal{A}$ √ **if** [*WT-intro*]: $\mathcal{I}$ ⊢*g* $\mathcal{A}$ √
⟨*proof*⟩

lemma *interaction-bounded-by-obsf-distinguisher* [*interaction-bound*]:
interaction-bounded-by consider (*obsf-distinguisher* $\mathcal{A}$) *bound*
if [*interaction-bound*]: *interaction-bounded-by consider* $\mathcal{A}$ *bound*

⟨*proof*⟩

lemma *plossless-obsf-distinguisher* [*simp*]:
plossless-gpv (*exception-ℐ ℐ*) (*obsf-distinguisher 𝒜*)
if *plossless-gpv ℐ 𝒜 ℐ* ⊢*g 𝒜* √
⟨*proof*⟩

type-synonym ('*a*, '*b*, '*c*, '*d*) *converter-obsf* = ('*a*, '*b exception*, '*c*, '*d exception*) *converter*

translations
(*type*) ('*a*, '*b*, '*c*, '*d*) *converter-obsf* <= (*type*) ('*a*, '*b exception*, '*c*, '*d exception*) *converter*

primcorec *obsf-converter* :: ('*a*, '*b*, '*c*, '*d*) *converter* ⇒ ('*a*, '*b*, '*c*, '*d*) *converter-obsf*
where
run-converter (*obsf-converter conv*) = (λ*x*.
map-gpv (*map-prod id obsf-converter*) *id*
(*map-gpv* (λ*convF*. *case convF of Fault* ⇒ (*Fault*, *fail-converter*) | *OK* (*a*, *conv'*) ⇒ (*OK a*, *conv'*)) *id*
(*try-gpv* (*map-gpv' exception-of-option id option-of-exception* (*gpv-stop* (*run-converter conv x*))) (*Done Fault*))))

lemma *obsf-converter-exception* [*simp*]: *obsf-converter fail-converter* = *const-converter Fault*
⟨*proof*⟩

lemma *obsf-converter-sel* [*simp*]:
run-converter (*obsf-converter conv*) *x* =
TRY map-gpv' (λ*y*. *case y of None* ⇒ (*Fault*, *const-converter Fault*) | *Some*(*x*, *conv'*) ⇒ (*OK x*, *obsf-converter conv'*)) *id option-of-exception*
(*gpv-stop* (*run-converter conv x*))
ELSE Done (*Fault*, *const-converter Fault*)
⟨*proof*⟩

declare *obsf-converter.sel* [*simp del*]

lemma *exec-gpv-obsf-resource*:
defines *exec-gpv1* ≡ *exec-gpv*
and *exec-gpv2* ≡ *exec-gpv*
shows
exec-gpv1 run-resource (*map-gpv' id id option-of-exception* (*gpv-stop gpv*)) (*obsf-resource res*) ↾ {(*Some x*, *y*)|*x y*. *True*} =
map-spmf (*map-prod Some obsf-resource*) (*exec-gpv2 run-resource gpv res*)
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *obsf-attach*:

assumes *pfinite*: *pfinite-converter* $\mathcal{I}$ $\mathcal{I}'$ *conv*
and *WT*: $\mathcal{I}$, $\mathcal{I}' \vdash_C$ *conv* $\surd$
and *WT-resource*: $\mathcal{I}' \vdash$*res res* $\surd$
shows *outs-*$\mathcal{I}$ $\mathcal{I} \vdash_R$ *attach* (*obsf-converter conv*) (*obsf-resource res*) $\sim$ *obsf-resource* (*attach conv res*)
⟨*proof*⟩

lemma *colossless-obsf-converter* [*simp*]:
colossless-converter (*exception-*$\mathcal{I}$ $\mathcal{I}$) $\mathcal{I}'$ (*obsf-converter conv*)
⟨*proof*⟩

lemma *WT-obsf-converter* [*WT-intro*]:
exception-$\mathcal{I}$ $\mathcal{I}$, *exception-*$\mathcal{I}$ $\mathcal{I}' \vdash_C$ *obsf-converter conv* $\surd$ **if** $\mathcal{I}$, $\mathcal{I}' \vdash_C$ *conv* $\surd$
⟨*proof*⟩

lemma *inline1-gpv-stop-obsf-converter*:
defines *inline1a* ≡ *inline1*
and *inline1b* ≡ *inline1*
shows *bind-spmf* (*inline1a run-converter* (*map-gpv′ id id option-of-exception* (*gpv-stop gpv*)) (*obsf-converter conv*))
(λ*xy. case xy of Inl* (*None*, *conv′*) ⇒ *return-pmf None* | *Inl* (*Some x*, *conv′*) ⇒ *return-spmf* (*Inl* (*x*, *conv′*)) | *Inr y* ⇒ *return-spmf* (*Inr y*)) =
map-spmf (*map-sum* (*apsnd obsf-converter*)
(*apsnd* (*map-prod* (λ*rpv input. case input of Fault* ⇒ *Done* (*Fault*, *const-converter Fault*) | *OK input′* ⇒
map-gpv′ (λ*res. case res of None* ⇒ (*Fault*, *const-converter Fault*) | *Some* (*x*, *conv′*) ⇒ (*OK x*, *obsf-converter conv′*)) *id option-of-exception* (*try-gpv* (*gpv-stop* (*rpv input′*)) (*Done None*)))
(λ*rpv input. case input of Fault* ⇒ *Done None* | *OK input′* ⇒ *map-gpv′ id id option-of-exception* (*gpv-stop* (*rpv input′*))))))
(*inline1b run-converter gpv conv*)
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *inline-gpv-stop-obsf-converter*:
bind-gpv (*inline run-converter* (*map-gpv′ id id option-of-exception* (*gpv-stop gpv*)) (*obsf-converter conv*)) (λ(*x*, *conv′*). *case x of None* ⇒ *Fail* | *Some x′* ⇒ *Done* (*x*, *conv′*)) =
bind-gpv (*map-gpv′ id id option-of-exception* (*gpv-stop* (*inline run-converter gpv conv*))) (λ*x. case x of None* ⇒ *Fail* | *Some* (*x′*, *conv*) ⇒ *Done* (*Some x′*, *obsf-converter conv*))
⟨*proof*⟩

lemma *obsf-comp-converter*:
assumes *WT*: $\mathcal{I}$, $\mathcal{I}' \vdash_C$ *conv1* $\surd$ $\mathcal{I}'$, $\mathcal{I}'' \vdash_C$ *conv2* $\surd$
and *pfinite1*: *pfinite-converter* $\mathcal{I}$ $\mathcal{I}'$ *conv1*
shows *exception-*$\mathcal{I}$ $\mathcal{I}$, *exception-*$\mathcal{I}$ $\mathcal{I}'' \vdash_C$ *obsf-converter* (*comp-converter conv1*

conv2) ∼ *comp-converter* (*obsf-converter conv1*) (*obsf-converter conv2*)
⟨*proof*⟩

lemma *resource-of-obsf-oracle-Fault* [*simp*]:
resource-of-oracle (*obsf-oracle oracle*) *Fault* = *const-resource Fault*
⟨*proof*⟩

lemma *obsf-resource-of-oracle* [*simp*]:
obsf-resource (*resource-of-oracle oracle s*) = *resource-of-oracle* (*obsf-oracle oracle*)
(*OK s*)
⟨*proof*⟩

lemma *trace-callee-eq-obsf-Fault* [*simp*]: *A* ⊢$_C$ *obsf-oracle callee1*(*Fault*) ≈ *obsf-oracle callee2*(*Fault*)
⟨*proof*⟩

lemma *obsf-resource-eq-*$\mathcal{I}$*-cong*: *A* ⊢$_R$ *obsf-resource res1* ∼ *obsf-resource res2* **if** *A* ⊢$_R$ *res1* ∼ *res2*
⟨*proof*⟩

lemma *trace-callee-eq-obsf-oracleI*:
assumes *trace-callee-eq callee1 callee2 A p q*
shows *trace-callee-eq* (*obsf-oracle callee1*) (*obsf-oracle callee2*) *A* (*try-spmf* (*map-spmf OK p*) (*return-spmf Fault*)) (*try-spmf* (*map-spmf OK q*) (*return-spmf Fault*))
⟨*proof*⟩

lemma *trace-callee-eq′-obsf-resourceI*:
assumes *A* ⊢$_C$ *callee1*(*s*) ≈ *callee2*(*s′*)
shows *A* ⊢$_C$ *obsf-oracle callee1*(*OK s*) ≈ *obsf-oracle callee2*(*OK s′*)
⟨*proof*⟩

lemma *trace-eq-obsf-resourceI*:
assumes *A* ⊢$_R$ *res1* ≈ *res2*
shows *A* ⊢$_R$ *obsf-resource res1* ≈ *obsf-resource res2*
⟨*proof*⟩

lemma *spmf-run-obsf-oracle-obsf-distinguisher* [*rule-format*]:
defines *eg1* ≡ *exec-gpv* **and** *eg2* ≡ *exec-gpv* **shows**
spmf (*map-spmf fst* (*eg1* (*obsf-oracle oracle*) (*obsf-distinguisher gpv*) (*OK s*))) *True* =
spmf (*map-spmf fst* (*eg2 oracle gpv s*)) *True*
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *spmf-obsf-distinguisher-obsf-resource-True*:
spmf (*connect-obsf* (*obsf-distinguisher* $\mathcal{A}$) (*obsf-resource res*)) *True* = *spmf* (*connect* $\mathcal{A}$ *res*) *True*
⟨*proof*⟩

lemma *advantage-obsf-distinguisher*:
advantage (*obsf-distinguisher* $\mathcal{A}$) (*obsf-resource ideal-resource*) (*obsf-resource real-resource*)
=
advantage $\mathcal{A}$ *ideal-resource real-resource*
⟨*proof*⟩

end
theory *Fold-Spmf*
imports
More-CC
begin

primrec (*transfer*)
foldl-spmf :: ('*b* ⇒ '*a* ⇒ '*b spmf*) ⇒ '*b spmf* ⇒ '*a list* ⇒ '*b spmf*
where
foldl-spmf-Nil: *foldl-spmf f p* [] = *p*
| *foldl-spmf-Cons*: *foldl-spmf f p* (*x* # *xs*) = *foldl-spmf f* (*bind-spmf p* (λ*a*. *f a x*))
xs

lemma *foldl-spmf-return-pmf-None* [*simp*]:
foldl-spmf f (*return-pmf None*) *xs* = *return-pmf None*
⟨*proof*⟩

lemma *foldl-spmf-bind-spmf*: *foldl-spmf f* (*bind-spmf p g*) *xs* = *bind-spmf p* (λ*a*.
foldl-spmf f (*g a*) *xs*)
⟨*proof*⟩

lemma *bind-foldl-spmf-return*:
bind-spmf p (λ*x*. *foldl-spmf f* (*return-spmf x*) *xs*) = *foldl-spmf f p xs*
⟨*proof*⟩

lemma *foldl-spmf-map* [*simp*]: *foldl-spmf f p* (*map g xs*) = *foldl-spmf* (*map-fun id*
(*map-fun g id*) *f*) *p xs*
⟨*proof*⟩

lemma *foldl-spmf-identity* [*simp*]: *foldl-spmf* (λ*s x*. *return-spmf s*) *p xs* = *p*
⟨*proof*⟩

lemma *foldl-spmf-conv-foldl*:
foldl-spmf (λ*s x*. *return-spmf* (*f s x*)) *p xs* = *map-spmf* (λ*s*. *foldl f s xs*) *p*
⟨*proof*⟩

lemma *foldl-spmf-Cons'*:
foldl-spmf f (*return-spmf a*) (*x* # *xs*) = *bind-spmf* (*f a x*) (λ*a'*. *foldl-spmf f*
(*return-spmf a'*) *xs*)
⟨*proof*⟩

lemma *foldl-spmf-append*: *foldl-spmf f p* (*xs* @ *ys*) = *foldl-spmf f* (*foldl-spmf f p*

xs) *ys*
⟨*proof*⟩

lemma
foldl-spmf-helper:
assumes $\bigwedge x.\ h\ (f\ x) = x$
assumes $\bigwedge x.\ f\ (h\ x) = x$
shows *foldl-spmf* ($\lambda a\ e.$ *map-spmf h* (*g* (*f a*) *e*)) *acc es* =
map-spmf h (*foldl-spmf g* (*map-spmf f acc*) *es*)
⟨*proof*⟩

lemma
foldl-spmf-helper2:
assumes $\bigwedge x\ y.\ p\ (f\ x\ y) = x$
assumes $\bigwedge x\ y.\ q\ (f\ x\ y) = y$
assumes $\bigwedge x.\ f\ (p\ x)\ (q\ x) = x$
shows *foldl-spmf* ($\lambda a\ e.$ *map-spmf* (*f* (*p a*)) (*g* (*q a*) *e*)) *acc es* =
bind-spmf acc ($\lambda acc'.$ *map-spmf* (*f* (*p acc'*)) (*foldl-spmf g* (*return-spmf* (*q acc'*)) *es*))
⟨*proof*⟩

lemma *foldl-pair-constl*: *foldl* ($\lambda s\ e.$ *map-prod* (λ-. *c*) ($\lambda r.$ *f r e*) *s*) (*c*, *sr*) *l* =
Pair c (*foldl* ($\lambda s\ e.$ *f s e*) *sr l*)
⟨*proof*⟩

lemma *foldl-spmf-pair-left*:
foldl-spmf ($\lambda(l,\ r)\ e.$ *map-spmf* ($\lambda l'.$ (l', *r*)) (*f l e*)) (*return-spmf* (*l*, *r*)) *es* =
map-spmf ($\lambda l'.$ (l', *r*)) (*foldl-spmf f* (*return-spmf l*) *es*)
⟨*proof*⟩

lemma *foldl-spmf-pair-left2*:
foldl-spmf ($\lambda(l,$ -$)\ e.$ *map-spmf* ($\lambda l'.$ (l', c')) (*f l e*)) (*return-spmf* (*l*, *c*)) *es* =
map-spmf ($\lambda l'.$ (l', *if es* = [] *then c else* c')) (*foldl-spmf f* (*return-spmf l*) *es*)
⟨*proof*⟩

lemma *foldl-pair-constr*: *foldl* ($\lambda s\ e.$ *map-prod* ($\lambda l.$ *f l e*) (λ-. *c*) *s*) (*sl*, *c*) *l* =
Pair (*foldl* ($\lambda s\ e.$ *f s e*) *sl l*) *c*
⟨*proof*⟩

lemma *foldl-spmf-pair-right*:
foldl-spmf ($\lambda(l,\ r)\ e.$ *map-spmf* ($\lambda r'.$ (*l*, r')) (*f r e*)) (*return-spmf* (*l*, *r*)) *es* =
map-spmf ($\lambda r'.$ (*l*, r')) (*foldl-spmf f* (*return-spmf r*) *es*)
⟨*proof*⟩

lemma *foldl-spmf-pair-right2*:
foldl-spmf ($\lambda(-,\ r)\ e.$ *map-spmf* ($\lambda r'.$ (c', r')) (*f r e*)) (*return-spmf* (*c*, *r*)) *es* =
map-spmf ($\lambda r'.$ (*if es* = [] *then c else* c', r')) (*foldl-spmf f* (*return-spmf r*) *es*)
⟨*proof*⟩

lemma *foldl-spmf-pair-right3*:
foldl-spmf (λ(*l*, *r*) *e*. *map-spmf* (*Pair* (*g e*)) (*f r e*)) (*return-spmf* (*l*, *r*)) *es* =
map-spmf (*Pair* (*if es* = [] *then l else g* (*last es*))) (*foldl-spmf f* (*return-spmf r*) *es*)
⟨*proof*⟩

lemma *foldl-pullout*: *bind-spmf f* (λ*x*. *bind-spmf* (*foldl-spmf g init* (*events x*)) (λ*y*. *h x y*)) =
bind-spmf (*bind-spmf f* (λ*x*. *foldl-spmf* (λ(*l*, *r*) *e*. *map-spmf* (*Pair l*) (*g r e*)) (*map-spmf* (*Pair x*) *init*) (*events x*)))
(λ(*x*, *y*). *h x y*) **for** *f g h init events*
⟨*proof*⟩

lemma *bind-foldl-spmf-pair-append*:
bind-spmf
(*foldl-spmf* (λ(*x*, *y*) *e*. *map-spmf* (*apfst* ((@) *x*)) (*f y e*)) (*return-spmf* (*a* @ *c*, *b*)) *es*)
(λ(*x*, *y*). *g x y*) =
bind-spmf
(*foldl-spmf* (λ(*x*, *y*) *e*. *map-spmf* (*apfst* ((@) *x*)) (*f y e*)) (*return-spmf* (*c*, *b*)) *es*)
(λ(*x*, *y*). *g* (*a* @ *x*) *y*)
⟨*proof*⟩

lemma *foldl-spmf-chain*:
(*foldl-spmf* (λ(*oevents*, *s-event*) *event*. *map-spmf* (*map-prod* ((@) *oevents*) *id*) (*fff s-event event*)) (*return-spmf* ([], *s-event*)) *ievents*)
⟜ (λ(*oevents*, *s-event*′). *foldl-spmf ggg* (*return-spmf s-core*) *oevents*
⟜ (λ*s-core*′. *return-spmf* (*f s-core*′ *s-event*′))) =
foldl-spmf (λ(*s-event*, *s-core*) *event*. *fff s-event event* ⟜ (λ(*oevents*, *s-event*′).
map-spmf (*Pair s-event*′) (*foldl-spmf ggg* (*return-spmf s-core*) *oevents*))) (*return-spmf* (*s-event*, *s-core*)) *ievents*
⟜ (λ(*s-event*′, *s-core*′). *return-spmf* (*f s-core*′ *s-event*′))
⟨*proof*⟩
primrec *pauses* :: ′*a list* ⇒ (*unit*, ′*a*, ′*b*) *gpv* **where**
pauses [] = *Done* ()
| *pauses* (*x* # *xs*) = *Pause x* (λ-. *pauses xs*)

lemma *WT-gpv-pauses* [*WT-intro*]:
I ⊢*g pauses xs* √ **if** *set xs* ⊆ *outs-I I*
⟨*proof*⟩

lemma *exec-gpv-pauses*:
exec-gpv callee (*pauses xs*) *s* =
map-spmf (*Pair* ()) (*foldl-spmf* (*map-fun id* (*map-fun id* (*map-spmf snd*)) *callee*) (*return-spmf s*) *xs*)
⟨*proof*⟩

end
theory *Fused-Resource* **imports**
Fold-Spmf
begin

context includes $\mathcal{I}$.*lifting* **begin**
lift-definition $e\mathcal{I} :: ('a, 'b)\ \mathcal{I} \Rightarrow ('a, 'b \times 'c)\ \mathcal{I}$ **is** $\lambda\mathcal{I}\ x.\ \mathcal{I}\ x \times UNIV$ ⟨*proof*⟩

lemma *outs-*$\mathcal{I}$*-e*$\mathcal{I}$[*simp*]: *outs-*$\mathcal{I}$ $(e\mathcal{I}\ \mathcal{I}) =$ *outs-*$\mathcal{I}$ $\mathcal{I}$
⟨*proof*⟩

lemma *responses-*$\mathcal{I}$*-e*$\mathcal{I}$ [*simp*]: *responses-*$\mathcal{I}$ $(e\mathcal{I}\ \mathcal{I})\ x =$ *responses-*$\mathcal{I}$ $\mathcal{I}\ x \times UNIV$
⟨*proof*⟩

lemma *e*$\mathcal{I}$*-map-*$\mathcal{I}$: $e\mathcal{I}$ (*map-*$\mathcal{I}$ *f g* $\mathcal{I}$) = *map-*$\mathcal{I}$ *f* (*apfst g*) ($e\mathcal{I}$ $\mathcal{I}$)
⟨*proof*⟩

lemma *e*$\mathcal{I}$*-inverse* [*simp*]: *map-*$\mathcal{I}$ *id fst* ($e\mathcal{I}$ $\mathcal{I}$) = $\mathcal{I}$
⟨*proof*⟩
end
lifting-update $\mathcal{I}$.*lifting*
lifting-forget $\mathcal{I}$.*lifting*

4 Fused Resource

4.1 Event Oracles – they generate events

type-synonym
('*state*, '*event*, '*input*, '*output*) *eoracle* = ('*state*, '*input*, '*output* × '*event list*) *oracle*′

definition
parallel-eoracle ::
('*s1*, '*e1*, '*i1*, '*o1*) *eoracle* ⇒ ('*s2*, '*e2*, '*i2*, '*o2*) *eoracle* ⇒
('*s1* × '*s2*, '*e1* + '*e2*, '*i1* + '*i2*, '*o1* + '*o2*) *eoracle*
where
parallel-eoracle eoracle1 eoracle2 state ≡
comp
(*map-spmf*
(*map-prod*
(*case-sum*
(*map-prod Inl* (*map Inl*))
(*map-prod Inr* (*map Inr*)))
id))
(*parallel-oracle eoracle1 eoracle2 state*)

definition
plus-eoracle ::

```
  ('s, 'e1, 'i1, 'o1) eoracle ⇒ ('s, 'e2, 'i2, 'o2) eoracle ⇒
    ('s, 'e1 + 'e2, 'i1 + 'i2, 'o1 + 'o2) eoracle
  where
    plus-eoracle eoracle1 eoracle2 state ≡
      comp
       (map-spmf
         (map-prod
           (case-sum
             (map-prod Inl (map Inl))
             (map-prod Inr (map Inr)))
           id))
       (plus-oracle eoracle1 eoracle2 state)

definition
  translate-eoracle ::
    ('s-event, 'e1, 'e2 list) oracle' ⇒ ('s-event × 's, 'e1, 'i, 'o) eoracle ⇒
      ('s-event × 's, 'e2, 'i, 'o) eoracle
  where
    translate-eoracle translator eoracle state inp ≡
      bind-spmf
       (eoracle state inp)
       (λ((out, e-in), s).
        let conc = (λ(es, st) e. map-spmf (map-prod ((@) es) id) (translator st e))
in do {
         (e-out, s-event) ← foldl-spmf conc (return-spmf ([], fst s)) e-in;
         return-spmf ((out, e-out), s-event, snd s)
       })
```

4.2 Event Handlers – they absorb (and silently handle) events

```
type-synonym
  ('state, 'event) handler = 'state ⇒ 'event ⇒ 'state spmf

fun
   parallel-handler :: ('s1, 'e1) handler ⇒ ('s2, 'e2) handler ⇒ ('s1 × 's2, 'e1 +
'e2) handler
  where
    parallel-handler left - s (Inl e1) = map-spmf (λs1'. (s1', snd s)) (left (fst s) e1)
  | parallel-handler - right s (Inr e2) = map-spmf (λs2'. (fst s, s2')) (right (snd s)
e2)

definition
  plus-handler :: ('s, 'e1) handler ⇒ ('s, 'e2) handler ⇒ ('s, 'e1 + 'e2) handler
  where
    plus-handler left right s ≡ case-sum (left s) (right s)

lemma parallel-handler-left:
  map-fun id (map-fun Inl id) (parallel-handler left right) =
    (λ(s-l, s-r) q. map-spmf (λs-l'. (s-l', s-r)) (left s-l q))
```

⟨*proof*⟩

lemma *parallel-handler-right*:
map-fun id (*map-fun Inr id*) (*parallel-handler left right*) =
(λ(*s-l*, *s-r*) *q*. *map-spmf* (λ*s-r'*. (*s-l*, *s-r'*)) (*right s-r q*))
⟨*proof*⟩

lemma *in-set-spmf-parallel-handler*:
s' ∈ *set-spmf* (*parallel-handler left right s x*) ⟷
(*case x of Inl e* ⇒ *fst s'* ∈ *set-spmf* (*left* (*fst s*) *e*) ∧ *snd s'* = *snd s*
| *Inr e* ⇒ *snd s'* ∈ *set-spmf* (*right* (*snd s*) *e*) ∧ *fst s'* = *fst s*)
⟨*proof*⟩

4.3 Fused Resource Construction

codatatype
('*s-core*, '*event*, '*iadv-core*, '*iusr-core*, '*oadv-core*, '*ousr-core*) *core* =
Core
(*cpoke*: ('*s-core*, '*event*) *handler*)
(*cfunc-adv*: ('*s-core*, '*iadv-core*, '*oadv-core*) *oracle'*)
(*cfunc-usr*: ('*s-core*, '*iusr-core*, '*ousr-core*) *oracle'*)

declare *core.sel-transfer*[*transfer-rule del*]
declare *core.ctr-transfer*[*transfer-rule del*]
declare *core.case-transfer*[*transfer-rule del*]

context
includes *lifting-syntax*
begin

inductive
rel-core'::
('*s-core* ⇒ '*s-core'* ⇒ *bool*) ⇒
('*event* ⇒ '*event'* ⇒ *bool*) ⇒
('*iadv-core* ⇒ '*iadv-core'* ⇒ *bool*) ⇒
('*iusr-core* ⇒ '*iusr-core'* ⇒ *bool*) ⇒
('*oadv-core* ⇒ '*oadv-core'* ⇒ *bool*) ⇒
('*ousr-core* ⇒ '*ousr-core'* ⇒ *bool*) ⇒
('*s-core*, '*event*, '*iadv-core*, '*iusr-core*, '*oadv-core*, '*ousr-core*) *core* ⇒
('*s-core'*, '*event'*, '*iadv-core'*, '*iusr-core'*, '*oadv-core'*, '*ousr-core'*) *core* ⇒ *bool*
for *S E IA IU OA OU*
where *rel-core' S E IA IU OA OU* (*Core cpoke cfunc-adv cfunc-usr*) (*Core cpoke'* *cfunc-adv'* *cfunc-usr'*)
if
(*S* ===> *E* ===> *rel-spmf S*) *cpoke cpoke'* **and**
(*S* ===> *IA* ===> *rel-spmf* (*rel-prod OA S*)) *cfunc-adv cfunc-adv'* **and**
(*S* ===> *IU* ===> *rel-spmf* (*rel-prod OU S*)) *cfunc-usr cfunc-usr'*
for *cpoke cfunc-adv cfunc-usr*

inductive-simps
rel-core′-simps [*simp*]:
rel-core′ S E IA IU OA OU (*Core cpoke′ cfunc-adv′ cfunc-usr′*) (*Core cpoke″ cfunc-adv″ cfunc-usr″*)

lemma
rel-core′-eq [*relator-eq*]:
rel-core′ (=) (=) (=) (=) (=) (=) = (=)
⟨*proof*⟩

lemma
rel-core′-mono [*relator-mono*]:
rel-core′ S E IA IU OA OU ≤ *rel-core′ S E′ IA′ IU′ OA′ OU′*
if *E′* ≤ *E IA′* ≤ *IA IU′* ≤ *IU OA* ≤ *OA′ OU* ≤ *OU′*
⟨*proof*⟩

lemma
cpoke-parametric [*transfer-rule*]:
(*rel-core′ S E IA IU OA OU* ===> *S* ===> *E* ===> *rel-spmf S*) *cpoke cpoke*
⟨*proof*⟩

lemma
cfunc-adv-parametric [*transfer-rule*]:
(*rel-core′ S E IA IU OA OU* ===> *S* ===> *IA* ===> *rel-spmf* (*rel-prod OA S*)) *cfunc-adv cfunc-adv*
⟨*proof*⟩

lemma
cfunc-usr-parametric [*transfer-rule*]:
(*rel-core′ S E IA IU OA OU* ===> *S* ===> *IU* ===> *rel-spmf* (*rel-prod OU S*)) *cfunc-usr cfunc-usr*
⟨*proof*⟩

lemma
Core-parametric [*transfer-rule*]:
((*S* ===> *E* ===> *rel-spmf S*) ===> (*S* ===> *IA* ===> *rel-spmf* (*rel-prod OA S*)) ===> (*S* ===> *IU* ===> *rel-spmf* (*rel-prod OU S*))
===> *rel-core′ S E IA IU OA OU*) *Core Core*
⟨*proof*⟩

lemma
case-core-parametric [*transfer-rule*]:
(((*S* ===> *E* ===> *rel-spmf S*) ===>
(*S* ===> *IA* ===> *rel-spmf* (*rel-prod OA S*)) ===>
(*S* ===> *IU* ===> *rel-spmf* (*rel-prod OU S*)) ===> *X*) ===>
rel-core′ S E IA IU OA OU ===> *X*) *case-core case-core*
⟨*proof*⟩

lemma
corec-core-parametric [*transfer-rule*]:
((*X* ===> *S* ===> *E* ===> *rel-spmf S*) ===>
(*X* ===> *S* ===> *IA* ===> *rel-spmf* (*rel-prod OA S*)) ===>
(*X* ===> *S* ===> *IU* ===> *rel-spmf* (*rel-prod OU S*)) ===>
X ===> *rel-core′ S E IA IU OA OU*) *corec-core corec-core*
⟨*proof*⟩

primcorec *map-core′* ::
('*event′* ⇒ '*event*) ⇒
('*iadv-core′* ⇒ '*iadv-core*) ⇒
('*iusr-core′* ⇒ '*iusr-core*) ⇒
('*oadv-core* ⇒ '*oadv-core′*) ⇒
('*ousr-core* ⇒ '*ousr-core′*) ⇒
('*s-core*, '*event*, '*iadv-core*, '*iusr-core*, '*oadv-core*, '*ousr-core*) *core* ⇒
('*s-core*, '*event′*, '*iadv-core′*, '*iusr-core′*, '*oadv-core′*, '*ousr-core′*) *core*
where
cpoke (*map-core′ e ia iu oa ou core*) = (*id* −−−> *e* −−−> *id*) (*cpoke core*)
| *cfunc-adv* (*map-core′ e ia iu oa ou core*) = (*id* −−−> *ia* −−−> *map-spmf* (*map-prod oa id*)) (*cfunc-adv core*)
| *cfunc-usr* (*map-core′ e ia iu oa ou core*) = (*id* −−−> *iu* −−−> *map-spmf* (*map-prod ou id*)) (*cfunc-usr core*)

lemmas *map-core′-simps* [*simp*] = *map-core′.ctr*[**where** *core*=*Core - - -*, *simplified*]

parametric-constant *map-core′-parametric*[*transfer-rule*]: *map-core′-def*

lemma *core′-rel-Grp*:
rel-core′ (=) (*BNF-Def.Grp UNIV e*)$^{-1-1}$ (*BNF-Def.Grp UNIV ia*)$^{-1-1}$ (*BNF-Def.Grp UNIV iu*)$^{-1-1}$ (*BNF-Def.Grp UNIV oa*) (*BNF-Def.Grp UNIV ou*)
= *BNF-Def.Grp UNIV* (*map-core′ e ia iu oa ou*)
⟨*proof*⟩

end

inductive *WT-core* :: ('*iadv*, '*oadv*) *I* ⇒ ('*iusr*, '*ousr*) *I* ⇒ ('*s* ⇒ *bool*) ⇒ ('*s*, '*event*, '*iadv*, '*iusr*, '*oadv*, '*ousr*) *core* ⇒ *bool*
for *I-adv I-usr I core* **where**
WT-core I-adv I-usr I core **if**
⋀*s e s′*. ⟦ *s′* ∈ *set-spmf* (*cpoke core s e*); *I s* ⟧ ⟹ *I s′*
⋀*s x y s′*. ⟦ (*y*, *s′*) ∈ *set-spmf* (*cfunc-adv core s x*); *x* ∈ *outs-I I-adv*; *I s* ⟧ ⟹ *y* ∈ *responses-I I-adv x* ∧ *I s′*
⋀*s x y s′*. ⟦ (*y*, *s′*) ∈ *set-spmf* (*cfunc-usr core s x*); *x* ∈ *outs-I I-usr*; *I s* ⟧ ⟹ *y* ∈ *responses-I I-usr x* ∧ *I s′*

lemma *WT-coreD*:
assumes *WT-core I-adv I-usr I core*
shows *WT-coreD-cpoke*: ⋀*s e s′*. ⟦ *s′* ∈ *set-spmf* (*cpoke core s e*); *I s* ⟧ ⟹ *I s′*
and *WT-coreD-cfunc-adv*: ⋀*s x y s′*. ⟦ (*y*, *s′*) ∈ *set-spmf* (*cfunc-adv core s x*);

$x \in$ *outs-$\mathcal{I}$ $\mathcal{I}$-adv*; *I s* $\rrbracket \Longrightarrow y \in$ *responses-$\mathcal{I}$ $\mathcal{I}$-adv x* $\wedge$ *I s'*
and *WT-coreD-cfund-usr*: $\bigwedge s\ x\ y\ s'.\ [\![(y,\ s') \in$ *set-spmf* (*cfunc-usr core s x*);
$x \in$ *outs-$\mathcal{I}$ $\mathcal{I}$-usr*; *I s* $\rrbracket \Longrightarrow y \in$ *responses-$\mathcal{I}$ $\mathcal{I}$-usr x* $\wedge$ *I s'*
⟨*proof*⟩

lemma *WT-coreD-foldl-spmf-cpoke*:
assumes *WT-core $\mathcal{I}$-adv $\mathcal{I}$-usr I core*
and *s'* $\in$ *set-spmf* (*foldl-spmf* (*cpoke core*) *p es*)
and $\forall$ *s* $\in$ *set-spmf p*. *I s*
shows *I s'*
⟨*proof*⟩

lemma *WT-core-trivial*:
assumes *adv*: $\bigwedge s$. *$\mathcal{I}$-adv* $\vdash c$ *cfunc-adv core s* $\surd$
and *usr*: $\bigwedge s$. *$\mathcal{I}$-usr* $\vdash c$ *cfunc-usr core s* $\surd$
shows *WT-core $\mathcal{I}$-adv $\mathcal{I}$-usr* (λ-. *True*) *core*
⟨*proof*⟩

codatatype
(*'s-rest*, *'event*, *'iadv-rest*, *'iusr-rest*, *'oadv-rest*, *'ousr-rest*, *'more*) *rest-scheme* =
Rest
(*rinit*: *'more*)
(*rfunc-adv*: (*'s-rest*, *'event*, *'iadv-rest*, *'oadv-rest*) *eoracle*)
(*rfunc-usr*: (*'s-rest*, *'event*, *'iusr-rest*, *'ousr-rest*) *eoracle*)

declare *rest-scheme.sel-transfer*[*transfer-rule del*]
declare *rest-scheme.ctr-transfer*[*transfer-rule del*]
declare *rest-scheme.case-transfer*[*transfer-rule del*]

context
includes *lifting-syntax*
begin

inductive
rel-rest'::
(*'s-rest* ⇒ *'s-rest'* ⇒ *bool*) ⇒
(*'event* ⇒ *'event'* ⇒ *bool*) ⇒
(*'iadv-rest* ⇒ *'iadv-rest'* ⇒ *bool*) ⇒
(*'iusr-rest* ⇒ *'iusr-rest'* ⇒ *bool*) ⇒
(*'oadv-rest* ⇒ *'oadv-rest'* ⇒ *bool*) ⇒
(*'ousr-rest* ⇒ *'ousr-rest'* ⇒ *bool*) ⇒
(*'more* ⇒ *'more'* ⇒ *bool*) ⇒
(*'s-rest*, *'event*, *'iadv-rest*, *'iusr-rest*, *'oadv-rest*, *'ousr-rest*, *'more*) *rest-scheme*
⇒
(*'s-rest'*, *'event'*, *'iadv-rest'*, *'iusr-rest'*, *'oadv-rest'*, *'ousr-rest'*, *'more'*) *rest-scheme*
⇒ *bool*
for *S E IA IU OA OU M*
where *rel-rest' S E IA IU OA OU M* (*Rest rinit rfunc-adv rfunc-usr*) (*Rest rinit'*
rfunc-adv' rfunc-usr')

if
M rinit rinit′ **and**
(*S ===> IA ===> rel-spmf* (*rel-prod* (*rel-prod OA* (*list-all2 E*)) *S*)) *rfunc-adv rfunc-adv′* **and**
(*S ===> IU ===> rel-spmf* (*rel-prod* (*rel-prod OU* (*list-all2 E*)) *S*)) *rfunc-usr rfunc-usr′*
for *rinit rfunc-adv rfunc-usr*

inductive-simps
rel-rest′-simps [*simp*]:
rel-rest′ S E IA IU OA OU M (*Rest rinit′ rfunc-adv′ rfunc-usr′*) (*Rest rinit″ rfunc-adv″ rfunc-usr″*)

lemma
rel-rest′-eq [*relator-eq*]:
rel-rest′ (=) (=) (=) (=) (=) (=) (=) = (=)
⟨*proof*⟩

lemma
rel-rest′-mono [*relator-mono*]:
rel-rest′ S E IA IU OA OU M ≤ *rel-rest′ S E′ IA′ IU′ OA′ OU′ M′*
if *E* ≤ *E′ IA′* ≤ *IA IU′* ≤ *IU OA* ≤ *OA′ OU* ≤ *OU′ M* ≤ *M′*
⟨*proof*⟩

lemma *rel-rest′-sel*: *rel-rest′ S E IA IU OA OU M rest1 rest2*
if *M* (*rinit rest1*) (*rinit rest2*)
and (*S ===> IA ===> rel-spmf* (*rel-prod* (*rel-prod OA* (*list-all2 E*)) *S*)) (*rfunc-adv rest1*) (*rfunc-adv rest2*)
and (*S ===> IU ===> rel-spmf* (*rel-prod* (*rel-prod OU* (*list-all2 E*)) *S*)) (*rfunc-usr rest1*) (*rfunc-usr rest2*)
⟨*proof*⟩

lemma *rinit-parametric* [*transfer-rule*]: (*rel-rest′ S E IA IU OA OU M ===> M*) *rinit rinit*
⟨*proof*⟩

lemma *rfunc-adv-parametric* [*transfer-rule*]:
(*rel-rest′ S E IA IU OA OU M ===> S ===> IA ===> rel-spmf* (*rel-prod* (*rel-prod OA* (*list-all2 E*)) *S*)) *rfunc-adv rfunc-adv*
⟨*proof*⟩

lemma *rfunc-usr-parametric* [*transfer-rule*]:
(*rel-rest′ S E IA IU OA OU M ===> S ===> IU ===> rel-spmf* (*rel-prod* (*rel-prod OU* (*list-all2 E*)) *S*)) *rfunc-usr rfunc-usr*
⟨*proof*⟩

lemma *Rest-parametric* [*transfer-rule*]:
(*M ===>* (*S ===> IA ===> rel-spmf* (*rel-prod* (*rel-prod OA* (*list-all2 E*)) *S*))

$===>$ $(S ===> IU ===> rel\text{-}spmf\ (rel\text{-}prod\ (rel\text{-}prod\ OU\ (list\text{-}all2\ E))\ S))$
$===>$ $rel\text{-}rest'\ S\ E\ IA\ IU\ OA\ OU\ M)\ Rest\ Rest$
⟨*proof*⟩

lemma *case-rest-scheme-parametric* [*transfer-rule*]:
$((M ===>$
$(S ===> IA ===> rel\text{-}spmf\ (rel\text{-}prod\ (rel\text{-}prod\ OA\ (list\text{-}all2\ E))\ S)) ===>$
$(S ===> IU ===> rel\text{-}spmf\ (rel\text{-}prod\ (rel\text{-}prod\ OU\ (list\text{-}all2\ E))\ S)) ===>$
$X) ===>$
$rel\text{-}rest'\ S\ E\ IA\ IU\ OA\ OU\ M ===> X)\ case\text{-}rest\text{-}scheme\ case\text{-}rest\text{-}scheme$
⟨*proof*⟩

lemma *corec-rest-scheme-parametric* [*transfer-rule*]:
$((X ===> M) ===>$
$(X ===> S ===> IA ===> rel\text{-}spmf\ (rel\text{-}prod\ (rel\text{-}prod\ OA\ (list\text{-}all2\ E))$
$S)) ===>$
$(X ===> S ===> IU ===> rel\text{-}spmf\ (rel\text{-}prod\ (rel\text{-}prod\ OU\ (list\text{-}all2\ E))$
$S)) ===>$
$X ===> rel\text{-}rest'\ S\ E\ IA\ IU\ OA\ OU\ M)\ corec\text{-}rest\text{-}scheme\ corec\text{-}rest\text{-}scheme$
⟨*proof*⟩

primcorec *map-rest'* ::
$('event \Rightarrow 'event') \Rightarrow$
$('iadv\text{-}rest' \Rightarrow 'iadv\text{-}rest) \Rightarrow$
$('iusr\text{-}rest' \Rightarrow 'iusr\text{-}rest) \Rightarrow$
$('oadv\text{-}rest \Rightarrow 'oadv\text{-}rest') \Rightarrow$
$('ousr\text{-}rest \Rightarrow 'ousr\text{-}rest') \Rightarrow$
$('more \Rightarrow 'more') \Rightarrow$
$('s\text{-}rest, 'event, 'iadv\text{-}rest, 'iusr\text{-}rest, 'oadv\text{-}rest, 'ousr\text{-}rest, 'more)\ rest\text{-}scheme$
$\Rightarrow$
$('s\text{-}rest, 'event', 'iadv\text{-}rest', 'iusr\text{-}rest', 'oadv\text{-}rest', 'ousr\text{-}rest', 'more')\ rest\text{-}scheme$
where
$rinit\ (map\text{-}rest'\ e\ ia\ iu\ oa\ ou\ m\ rest) = m\ (rinit\ rest)$
$|\ rfunc\text{-}adv\ (map\text{-}rest'\ e\ ia\ iu\ oa\ ou\ m\ rest) =$
$(id ---> ia ---> map\text{-}spmf\ (map\text{-}prod\ (map\text{-}prod\ oa\ (map\ e))\ id))\ (rfunc\text{-}adv$
$rest)$
$|\ rfunc\text{-}usr\ (map\text{-}rest'\ e\ ia\ iu\ oa\ ou\ m\ rest) =$
$(id ---> iu ---> map\text{-}spmf\ (map\text{-}prod\ (map\text{-}prod\ ou\ (map\ e))\ id))\ (rfunc\text{-}usr$
$rest)$

lemmas *map-rest'-simps* [*simp*] = *map-rest'.ctr*[**where** *rest=Rest - - -*, *simplified*]

parametric-constant *map-rest'-parametric*[*transfer-rule*]: *map-rest'-def*

lemma *rest'-rel-Grp*:
$rel\text{-}rest'\ (=)\ (BNF\text{-}Def.Grp\ UNIV\ e)\ (BNF\text{-}Def.Grp\ UNIV\ ia)^{-1-1}\ (BNF\text{-}Def.Grp$
$UNIV\ iu)^{-1-1}\ (BNF\text{-}Def.Grp\ UNIV\ oa)\ (BNF\text{-}Def.Grp\ UNIV\ ou)\ (BNF\text{-}Def.Grp$
$UNIV\ m)$
$= BNF\text{-}Def.Grp\ UNIV\ (map\text{-}rest'\ e\ ia\ iu\ oa\ ou\ m)$

⟨*proof*⟩

end

type-synonym
(*'s-rest*, *'event*, *'iadv-rest*, *'iusr-rest*, *'oadv-rest*, *'ousr-rest*) *rest-wstate* =
(*'s-rest*, *'event*, *'iadv-rest*, *'iusr-rest*, *'oadv-rest*, *'ousr-rest*, *'s-rest*) *rest-scheme*

inductive *WT-rest* :: (*'iadv*, *'oadv*) $\mathcal{I}$ ⇒ (*'iusr*, *'ousr*) $\mathcal{I}$ ⇒ (*'s* ⇒ *bool*) ⇒ (*'s*, *'event*, *'iadv*, *'iusr*, *'oadv*, *'ousr*) *rest-wstate* ⇒ *bool*
for $\mathcal{I}$-*adv* $\mathcal{I}$-*usr I rest* **where**
WT-rest $\mathcal{I}$-*adv* $\mathcal{I}$-*usr I rest* **if**
⋀*s x y es s'*. ⟦ ((*y*, *es*), *s'*) ∈ *set-spmf* (*rfunc-adv rest s x*); *x* ∈ *outs-*$\mathcal{I}$ $\mathcal{I}$-*adv*; *I s* ⟧ ⟹ *y* ∈ *responses-*$\mathcal{I}$ $\mathcal{I}$-*adv x* ∧ *I s'*
⋀*s x y es s'*. ⟦ ((*y*, *es*), *s'*) ∈ *set-spmf* (*rfunc-usr rest s x*); *x* ∈ *outs-*$\mathcal{I}$ $\mathcal{I}$-*usr*; *I s* ⟧ ⟹ *y* ∈ *responses-*$\mathcal{I}$ $\mathcal{I}$-*usr x* ∧ *I s'*
I (*rinit rest*)

lemma *WT-restD*:
assumes *WT-rest* $\mathcal{I}$-*adv* $\mathcal{I}$-*usr I rest*
shows *WT-restD-rfunc-adv*: ⋀*s x y es s'*. ⟦ ((*y*, *es*), *s'*) ∈ *set-spmf* (*rfunc-adv rest s x*); *x* ∈ *outs-*$\mathcal{I}$ $\mathcal{I}$-*adv*; *I s* ⟧ ⟹ *y* ∈ *responses-*$\mathcal{I}$ $\mathcal{I}$-*adv x* ∧ *I s'*
and *WT-restD-rfunc-usr*: ⋀*s x y es s'*. ⟦ ((*y*, *es*), *s'*) ∈ *set-spmf* (*rfunc-usr rest s x*); *x* ∈ *outs-*$\mathcal{I}$ $\mathcal{I}$-*usr*; *I s* ⟧ ⟹ *y* ∈ *responses-*$\mathcal{I}$ $\mathcal{I}$-*usr x* ∧ *I s'*
and *WT-restD-rinit*: *I* (*rinit rest*)
⟨*proof*⟩

abbreviation
fuse-cfunc ::
(*'o* ⇒ *'x*) ⇒ (*'s-core*, *'i*, *'o*) *oracle'* ⇒ (*'s-core* × *'s-rest*, *'i* , *'x*) *oracle'*
where
fuse-cfunc redirect cfunc state inp ≡ *do* {
let handle = *map-prod redirect* (*prod.swap o Pair* (*snd state*));
(*os-cfunc* :: *'o* × *'s-core*) ← *cfunc* (*fst state*) *inp*;
return-spmf (*handle os-cfunc*)
}

abbreviation
fuse-rfunc ::
(*'o* ⇒ *'x*) ⇒ (*'s-rest*, *'e*, *'i*, *'o*) *eoracle* ⇒ (*'s-core*, *'e*) *handler* ⇒
(*'s-core* × *'s-rest*, *'i* , *'x*) *oracle'*
where
fuse-rfunc redirect rfunc notify state inp ≡
bind-spmf
(*rfunc* (*snd state*) *inp*)
(λ((*o-rfunc*, *e-lst*), *s-rfunc*).
bind-spmf
(*foldl-spmf notify* (*return-spmf* (*fst state*)) *e-lst*)
(λ*s-notify*. *return-spmf* (*redirect o-rfunc*, *s-notify*, *s-rfunc*)))

locale *fused-resource* =
  fixes
    core :: (*'s-core*, *'event*, *'iadv-core*, *'iusr-core*, *'oadv-core*, *'ousr-core*) *core* **and**
    core-init :: *'s-core*
begin

fun
  fuse ::
    (*'s-rest*, *'event*, *'iadv-rest*, *'iusr-rest*, *'oadv-rest*, *'ousr-rest*, *'m*) *rest-scheme* ⇒
    (*'s-core* × *'s-rest*,
      (*'iadv-core* + *'iadv-rest*) + (*'iusr-core* + *'iusr-rest*),
      (*'oadv-core* + *'oadv-rest*) + (*'ousr-core* + *'ousr-rest*)) *oracle'*
  where
    fuse rest state (*Inl* (*Inl iadv-core*)) =
      fuse-cfunc (*Inl o Inl*) (*cfunc-adv core*) *state iadv-core*
  | *fuse rest state* (*Inl* (*Inr iadv-rest*)) =
      fuse-rfunc (*Inl o Inr*) (*rfunc-adv rest*) (*cpoke core*) *state iadv-rest*
  | *fuse rest state* (*Inr* (*Inl iusr-core*)) =
      fuse-cfunc (*Inr o Inl*) (*cfunc-usr core*) *state iusr-core*
  | *fuse rest state* (*Inr* (*Inr iusr-rest*)) =
      fuse-rfunc (*Inr o Inr*) (*rfunc-usr rest*) (*cpoke core*) *state iusr-rest*

case-of-simps *fuse-case*: *fused-resource.fuse.simps*

lemma *callee-invariant-on-fuse*:
  assumes *WT-core* $\mathcal{I}$*-adv-core* $\mathcal{I}$*-usr-core I-core core*
    and *WT-rest* $\mathcal{I}$*-adv-rest* $\mathcal{I}$*-usr-rest I-rest rest*
  shows *callee-invariant-on* (*fuse rest*) (*pred-prod I-core I-rest*) ((*I-adv-core* $\oplus_{\mathcal{I}}$ $\mathcal{I}$*-adv-rest*) $\oplus_{\mathcal{I}}$ (*I-usr-core* $\oplus_{\mathcal{I}}$ $\mathcal{I}$*-usr-rest*))
⟨*proof*⟩

definition
  resource ::
    (*'s-rest*, *'event*, *'iadv-rest*, *'iusr-rest*, *'oadv-rest*, *'ousr-rest*) *rest-wstate* ⇒
    ((*'iadv-core* + *'iadv-rest*) + (*'iusr-core* + *'iusr-rest*),
      (*'oadv-core* + *'oadv-rest*) + (*'ousr-core* + *'ousr-rest*)) *resource*
  where
    resource rest = *resource-of-oracle* (*fuse rest*) (*core-init*, *rinit rest*)

lemma *WT-resource* [*WT-intro*]:
  assumes *WT-core* $\mathcal{I}$*-adv-core* $\mathcal{I}$*-usr-core I-core core*
    and *WT-rest* $\mathcal{I}$*-adv-rest* $\mathcal{I}$*-usr-rest I-rest rest*
    and *I-core core-init*
  shows (*I-adv-core* $\oplus_{\mathcal{I}}$ $\mathcal{I}$*-adv-rest*) $\oplus_{\mathcal{I}}$ (*I-usr-core* $\oplus_{\mathcal{I}}$ $\mathcal{I}$*-usr-rest*) ⊢*res resource rest* √
⟨*proof*⟩

end

parametric-constant
fuse-parametric [*transfer-rule*]: *fused-resource.fuse-case*

4.4 More helpful construction functions

context
fixes
core1 :: ('*s-core1*, '*event1*, '*iadv-core1*, '*iusr-core1*, '*oadv-core1*, '*ousr-core1*) *core*
and
core2 :: ('*s-core2*, '*event2*, '*iadv-core2*, '*iusr-core2*, '*oadv-core2*, '*ousr-core2*) *core*
begin

primcorec *parallel-core* ::
('*s-core1* × '*s-core2*, '*event1* + '*event2*,
'*iadv-core1* + '*iadv-core2*, '*iusr-core1* + '*iusr-core2*,
'*oadv-core1* + '*oadv-core2*, '*ousr-core1* + '*ousr-core2*) *core*
where
cpoke parallel-core = *parallel-handler* (*cpoke core1*) (*cpoke core2*)
| *cfunc-adv parallel-core* = *parallel-oracle* (*cfunc-adv core1*) (*cfunc-adv core2*)
| *cfunc-usr parallel-core* = *parallel-oracle* (*cfunc-usr core1*) (*cfunc-usr core2*)

end

context
fixes
cnv-adv :: '*s-adv* ⇒ '*iadv* ⇒ ('*oadv* × '*s-adv*, '*iadv-core*, '*oadv-core*) *gpv* **and**
cnv-usr :: '*s-usr* ⇒ '*iusr* ⇒ ('*ousr* × '*s-usr*, '*iusr-core*, '*ousr-core*) *gpv* **and**
core :: ('*s-core*, '*event*, '*iadv-core*, '*iusr-core*, '*oadv-core*, '*ousr-core*) *core*
begin

primcorec
attach-core :: (('*s-adv* × '*s-usr*) × '*s-core*, '*event*, '*iadv*, '*iusr*, '*oadv*, '*ousr*) *core*
where
cpoke attach-core = (λ(*s-advusr*, *s-core*) *event*.
map-spmf (λ*s-core*′. (*s-advusr*, *s-core*′)) (*cpoke core s-core event*))
| *cfunc-adv attach-core* = (λ((*s-adv*, *s-usr*), *s-core*) *iadv*.
map-spmf
(λ((*oadv*, *s-adv*′), *s-core*′). (*oadv*, ((*s-adv*′, *s-usr*), *s-core*′)))
(*exec-gpv* (*cfunc-adv core*) (*cnv-adv s-adv iadv*) *s-core*))
| *cfunc-usr attach-core* = (λ((*s-adv*, *s-usr*), *s-core*) *iusr*.
map-spmf
(λ((*ousr*, *s-usr*′), *s-core*′). (*ousr*, ((*s-adv*, *s-usr*′), *s-core*′)))
(*exec-gpv* (*cfunc-usr core*) (*cnv-usr s-usr iusr*) *s-core*))

end

lemma
attach-core-id-oracle-adv: *cfunc-adv* (*attach-core* 1_I *cnv core*) =
(λ(*s-cnv*, *s-core*) *q*. *map-spmf* (λ(*out*, *s-core'*). (*out*, *s-cnv*, *s-core'*)) (*cfunc-adv core s-core q*))
⟨*proof*⟩

lemma
attach-core-id-oracle-usr: *cfunc-usr* (*attach-core cnv* 1_I *core*) =
(λ(*s-cnv*, *s-core*) *q*. *map-spmf* (λ(*out*, *s-core'*). (*out*, *s-cnv*, *s-core'*)) (*cfunc-usr core s-core q*))
⟨*proof*⟩

context
fixes
rest1 :: ('*s-rest1*, '*event1*, '*iadv-rest1*, '*iusr-rest1*, '*oadv-rest1*, '*ousr-rest1*, '*more1*) *rest-scheme* **and**
rest2 :: ('*s-rest2*, '*event2*, '*iadv-rest2*, '*iusr-rest2*, '*oadv-rest2*, '*ousr-rest2*, '*more2*) *rest-scheme*
begin

primcorec *parallel-rest* ::
('*s-rest1* × '*s-rest2*, '*event1* + '*event2*, '*iadv-rest1* + '*iadv-rest2*, '*iusr-rest1* + '*iusr-rest2*,
'*oadv-rest1* + '*oadv-rest2*, '*ousr-rest1* + '*ousr-rest2*, '*more1* × '*more2*) *rest-scheme*

where
rinit parallel-rest = (*rinit rest1*, *rinit rest2*)
| *rfunc-adv parallel-rest* = *parallel-eoracle* (*rfunc-adv rest1*) (*rfunc-adv rest2*)
| *rfunc-usr parallel-rest* = *parallel-eoracle* (*rfunc-usr rest1*) (*rfunc-usr rest2*)

end

lemma *WT-parallel-rest* [*WT-intro*]:
WT-rest (*I-adv1* $\oplus_{\mathcal{I}}$ *I-adv2*) (*I-usr1* $\oplus_{\mathcal{I}}$ *I-usr2*) (*pred-prod I1 I2*) (*parallel-rest rest1 rest2*)
if *WT-rest I-adv1 I-usr1 I1 rest1*
and *WT-rest I-adv2 I-usr2 I2 rest2*
⟨*proof*⟩

context
fixes
cnv-adv :: '*s-adv* ⇒ '*iadv* ⇒ ('*oadv* × '*s-adv*, '*iadv-rest*, '*oadv-rest*) *gpv* **and**
cnv-usr :: '*s-usr* ⇒ '*iusr* ⇒ ('*ousr* × '*s-usr*, '*iusr-rest*, '*ousr-rest*) *gpv* **and**
f-init :: '*more* ⇒ '*more'* **and**
rest :: ('*s-rest*, '*event*, '*iadv-rest*, '*iusr-rest*, '*oadv-rest*, '*ousr-rest*, '*more*) *rest-scheme*
begin

primcorec
attach-rest ::
(('*s-adv* × '*s-usr*) × '*s-rest*, '*event*, '*iadv*, '*iusr*, '*oadv*, '*ousr*, '*more*') *rest-scheme*
where
rinit attach-rest = *f-init* (*rinit rest*)
| *rfunc-adv attach-rest* = (λ((*s-adv*, *s-usr*), *s-rest*) *iadv*.
let orc-of = λ*orc* (*s*, *es*) *q*. *map-spmf* (λ ((*out*, *e*), *s'*). (*out*, *s'*, *es* @ *e*)) (*orc s q*) *in*
let eorc-of = λ((*oadv*, *s-adv'*), (*s-rest'*, *es*)). ((*oadv*, *es*), ((*s-adv'*, *s-usr*), *s-rest'*)) *in*
map-spmf eorc-of (*exec-gpv* (*orc-of* (*rfunc-adv rest*)) (*cnv-adv s-adv iadv*) (*s-rest*, [])))
| *rfunc-usr attach-rest* = (λ((*s-adv*, *s-usr*), *s-rest*) *iusr*.
let orc-of = λ*orc* (*s*, *es*) *q*. *map-spmf* (λ ((*out*, *e*), *s'*). (*out*, *s'*, *es* @ *e*)) (*orc s q*) *in*
let eorc-of = λ((*ousr*, *s-usr'*), (*s-rest'*, *es*)). ((*ousr*, *es*), ((*s-adv*, *s-usr'*), *s-rest'*)) *in*
map-spmf eorc-of (*exec-gpv* (*orc-of* (*rfunc-usr rest*)) (*cnv-usr s-usr iusr*) (*s-rest*, [])))

end

lemma
attach-rest-id-oracle-adv: *rfunc-adv* (*attach-rest* 1_I *cnv f-init rest*) =
(λ(*s-cnv*, *s-core*) *q*. *map-spmf* (λ(*out*, *s-core'*). (*out*, *s-cnv*, *s-core'*)) (*rfunc-adv rest s-core q*))
⟨*proof*⟩

lemma
attach-rest-id-oracle-usr: *rfunc-usr* (*attach-rest cnv* 1_I *f-init rest*) =
(λ(*s-cnv*, *s-core*) *q*. *map-spmf* (λ(*out*, *s-core'*). (*out*, *s-cnv*, *s-core'*)) (*rfunc-usr rest s-core q*))
⟨*proof*⟩

5 Traces

type-synonym ('*event*, '*iadv-core*, '*iusr-core*, '*oadv-core*, '*ousr-core*) *trace-core* =
('*event* + '*iadv-core* × '*oadv-core* + '*iusr-core* × '*ousr-core*) *list*
⇒ ('*event* ⇒ *real*)
× ('*iadv-core* ⇒ '*oadv-core spmf*)
× ('*iusr-core* ⇒ '*ousr-core spmf*)

context
fixes *core* :: ('*s-core*, '*event*, '*iadv-core*, '*iusr-core*, '*oadv-core*, '*ousr-core*) *core*
begin

primrec *trace-core'* :: '*s-core spmf* ⇒ ('*event*, '*iadv-core*, '*iusr-core*, '*oadv-core*,

'ousr-core) trace-core where
trace-core' S [] =
(λe. weight-spmf' (bind-spmf S (λs. cpoke core s e)),
λia. bind-spmf S (λs. map-spmf fst (cfunc-adv core s ia)),
λiu. bind-spmf S (λs. map-spmf fst (cfunc-usr core s iu)))
| trace-core' S (obs # tr) = (case obs of
Inl e ⇒ trace-core' (mk-lossless (bind-spmf S (λs. cpoke core s e))) tr
| Inr (Inl (ia, oa)) ⇒ trace-core' (cond-spmf-fst (bind-spmf S (λs. cfunc-adv core s ia)) oa) tr
| Inr (Inr (iu, ou)) ⇒ trace-core' (cond-spmf-fst (bind-spmf S (λs. cfunc-usr core s iu)) ou) tr
)

end

declare *trace-core'.simps* [*simp del*]
case-of-simps *trace-core'-unfold*: *trace-core'.simps*[*unfolded weight-spmf'-def*]
simps-of-case *trace-core'-simps* [*simp*]: *trace-core'-unfold*

context includes *lifting-syntax* **begin**

lemma *trace-core'-parametric* [*transfer-rule*]:
(*rel-core' S E IA IU* (=) (=) ===>
rel-spmf S ===>
list-all2 (*rel-sum E* (*rel-sum* (*rel-prod IA* (=)) (*rel-prod IU* (=)))) ===>
rel-prod (*E* ===> (=)) (*rel-prod* (*IA* ===> (=)) (*IU* ===> (=))))
trace-core' trace-core'
⟨*proof*⟩

definition *trace-core-eq*
:: ('*s-core*, '*event*, '*iadv-core*, '*iusr-core*, '*oadv-core*, '*ousr-core*) *core*
⇒ ('*s-core'*, '*event*, '*iadv-core*, '*iusr-core*, '*oadv-core*, '*ousr-core*) *core*
⇒ '*event set* ⇒ '*iadv-core set* ⇒ '*iusr-core set*
⇒ '*s-core spmf* ⇒ '*s-core'* *spmf* ⇒ *bool* **where**
trace-core-eq core1 core2 E IA IU p q ⟷
(∀ *tr*. *set tr* ⊆ *E* <+> (*IA* × *UNIV*) <+> (*IU* × *UNIV*) ⟶
rel-prod (*eq-onp* (λe. *e* ∈ *E*) ===> (=)) (*rel-prod* (*eq-onp* (λia. *ia* ∈ *IA*) ===> (=)) (*eq-onp* (λiu. *iu* ∈ *IU*) ===> (=)))
(*trace-core' core1 p tr*) (*trace-core' core2 q tr*))

end

lemma *trace-core-eqD*:
assumes *trace-core-eq core1 core2 E IA IU p q*
and *set tr* ⊆ *E* <+> (*IA* × *UNIV*) <+> (*IU* × *UNIV*)
shows *trace-core-eqD-cpoke*:
e ∈ *E* ⟹ *fst* (*trace-core' core1 p tr*) *e* = *fst* (*trace-core' core2 q tr*) *e*
and *trace-core-eqD-cfunc-adv*:
ia ∈ *IA* ⟹ *fst* (*snd* (*trace-core' core1 p tr*)) *ia* = *fst* (*snd* (*trace-core' core2*

q tr)) *ia*
and *trace-core-eqD-cfunc-usr*:
iu ∈ *IU* ⟹ *snd* (*snd* (*trace-core′ core1 p tr*)) *iu* = *snd* (*snd* (*trace-core′ core2 q tr*)) *iu*
⟨*proof*⟩

lemma *trace-core-eqI*:
assumes ⋀*tr e*. ⟦ *set tr* ⊆ *E* <+> (*IA* × *UNIV*) <+> (*IU* × *UNIV*); *e* ∈ *E* ⟧
⟹ *fst* (*trace-core′ core1 p tr*) *e* = *fst* (*trace-core′ core2 q tr*) *e*
and ⋀*tr ia*. ⟦ *set tr* ⊆ *E* <+> (*IA* × *UNIV*) <+> (*IU* × *UNIV*); *ia* ∈ *IA* ⟧
⟹ *fst* (*snd* (*trace-core′ core1 p tr*)) *ia* = *fst* (*snd* (*trace-core′ core2 q tr*)) *ia*
and ⋀*tr iu*. ⟦ *set tr* ⊆ *E* <+> (*IA* × *UNIV*) <+> (*IU* × *UNIV*); *iu* ∈ *IU* ⟧
⟹ *snd* (*snd* (*trace-core′ core1 p tr*)) *iu* = *snd* (*snd* (*trace-core′ core2 q tr*)) *iu*
shows *trace-core-eq core1 core2 E IA IU p q*
⟨*proof*⟩

lemma *trace-core-return-pmf-None* [*simp*]:
trace-core′ core (*return-pmf None*) *tr* = (λ-. *0*, λ-. *return-pmf None*, λ-. *return-pmf None*)
⟨*proof*⟩

lemma *rel-core′-into-trace-core-eq*: *trace-core-eq core core′ E IA IU p q*
if *rel-core′ S* (*eq-onp* (λ*e*. *e* ∈ *E*)) (*eq-onp* (λ*ia*. *ia* ∈ *IA*)) (*eq-onp* (λ*iu*. *iu* ∈ *IU*)) (=) (=) *core core′*
rel-spmf S p q
⟨*proof*⟩

lemma *trace-core-eq-simI*:
fixes *core1* :: (*'s-core*, *'event*, *'iadv-core*, *'iusr-core*, *'oadv-core*, *'ousr-core*) *core*
and *core2* :: (*'s-core′*, *'event*, *'iadv-core*, *'iusr-core*, *'oadv-core*, *'ousr-core*) *core*
and *S* :: *'s-core spmf* ⇒ *'s-core′ spmf* ⇒ *bool*
assumes *start*: *S p q*
and *step-cpoke*: ⋀*p q e*. ⟦ *S p q*; *e* ∈ *E* ⟧ ⟹
weight-spmf (*bind-spmf p* (λ*s*. *cpoke core1 s e*)) = *weight-spmf* (*bind-spmf q* (λ*s*. *cpoke core2 s e*))
and *sim-cpoke*: ⋀*p q e*. ⟦ *S p q*; *e* ∈ *E* ⟧ ⟹
S (*mk-lossless* (*bind-spmf p* (λ*s*. *cpoke core1 s e*))) (*mk-lossless* (*bind-spmf q* (λ*s*. *cpoke core2 s e*)))
and *step-cfunc-adv*: ⋀*p q ia*. ⟦ *S p q*; *ia* ∈ *IA* ⟧ ⟹
bind-spmf p (λ*s1*. *map-spmf fst* (*cfunc-adv core1 s1 ia*)) = *bind-spmf q* (λ*s2*. *map-spmf fst* (*cfunc-adv core2 s2 ia*))
and *sim-cfunc-adv*: ⋀*p q ia s1 s2 s1′ s2′ oa*. ⟦ *S p q*; *ia* ∈ *IA*;
s1 ∈ *set-spmf p*; *s2* ∈ *set-spmf q*; (*oa*, *s1′*) ∈ *set-spmf* (*cfunc-adv core1 s1 ia*); (*oa*, *s2′*) ∈ *set-spmf* (*cfunc-adv core2 s2 ia*) ⟧
⟹ *S* (*cond-spmf-fst* (*bind-spmf p* (λ*s1*. *cfunc-adv core1 s1 ia*)) *oa*) (*cond-spmf-fst* (*bind-spmf q* (λ*s2*. *cfunc-adv core2 s2 ia*)) *oa*)
and *step-cfunc-usr*: ⋀*p q iu*. ⟦ *S p q*; *iu* ∈ *IU* ⟧ ⟹
bind-spmf p (λ*s1*. *map-spmf fst* (*cfunc-usr core1 s1 iu*)) = *bind-spmf q* (λ*s2*. *map-spmf fst* (*cfunc-usr core2 s2 iu*))

and *sim-cfunc-usr*: $\bigwedge p\ q\ iu\ s1\ s2\ s1'\ s2'\ ou.\ [\![\ S\ p\ q;\ iu \in IU;$
$s1 \in$ *set-spmf* $p;\ s2 \in$ *set-spmf* $q;\ (ou,\ s1') \in$ *set-spmf* (*cfunc-usr core1 s1 iu*);
$(ou,\ s2') \in$ *set-spmf* (*cfunc-usr core2 s2 iu*) $]\!]$
$\Longrightarrow S$ (*cond-spmf-fst* (*bind-spmf p* ($\lambda s1$. *cfunc-usr core1 s1 iu*)) *ou*) (*cond-spmf-fst* (*bind-spmf q* ($\lambda s2$. *cfunc-usr core2 s2 iu*)) *ou*)
shows *trace-core-eq core1 core2 E IA IU p q*
⟨*proof*⟩

context
fixes *core* :: (*'s-core*, *'event*, *'iadv-core*, *'iusr-core*, *'oadv-core*, *'ousr-core*) *core*
begin

fun *trace-core-aux*
:: *'s-core spmf* ⇒ (*'event* + *'iadv-core* × *'oadv-core* + *'iusr-core* × *'ousr-core*) *list*
⇒ *'s-core spmf* **where**
trace-core-aux p [] = *p*
| *trace-core-aux p* (*Inl e* # *tr*) = *trace-core-aux* (*mk-lossless* (*bind-spmf p* (λ*s*. *cpoke core s e*))) *tr*
| *trace-core-aux p* (*Inr* (*Inl* (*ia*, *oa*)) # *tr*) = *trace-core-aux* (*cond-spmf-fst* (*bind-spmf p* (λ*s*. *cfunc-adv core s ia*)) *oa*) *tr*
| *trace-core-aux p* (*Inr* (*Inr* (*iu*, *ou*)) # *tr*) = *trace-core-aux* (*cond-spmf-fst* (*bind-spmf p* (λ*s*. *cfunc-usr core s iu*)) *ou*) *tr*

end

lemma *trace-core-conv-trace-core-aux*:
trace-core' core p tr =
(λ*e*. *weight-spmf* (*bind-spmf* (*trace-core-aux core p tr*) (λ*s*. *cpoke core s e*)),
λ*ia*. *bind-spmf* (*trace-core-aux core p tr*) (λ*s*. *map-spmf fst* (*cfunc-adv core s ia*)),
λ*iu*. *bind-spmf* (*trace-core-aux core p tr*) (λ*s*. *map-spmf fst* (*cfunc-usr core s iu*)))
⟨*proof*⟩

lemma *trace-core-aux-append*:
trace-core-aux core p (*tr* @ *tr'*) = *trace-core-aux core* (*trace-core-aux core p tr*) *tr'*
⟨*proof*⟩

inductive *trace-core-closure*
:: (*'s-core*, *'event*, *'iadv-core*, *'iusr-core*, *'oadv-core*, *'ousr-core*) *core*
⇒ (*'s-core'*, *'event*, *'iadv-core*, *'iusr-core*, *'oadv-core*, *'ousr-core*) *core*
⇒ *'event set* ⇒ *'iadv-core set* ⇒ *'iusr-core set*
⇒ *'s-core spmf* ⇒ *'s-core' spmf* ⇒ *'s-core spmf* ⇒ *'s-core' spmf* ⇒ *bool*
for *core1 core2 E IA IU p q* **where**
trace-core-closure core1 core2 E IA IU p q (*trace-core-aux core1 p tr*) (*trace-core-aux core2 q tr*)
if *set tr* ⊆ *E* <+> *IA* × *UNIV* <+> *IU* × *UNIV*

lemma *trace-core-closure-start*: *trace-core-closure core1 core2 E IA IU p q p q*
⟨*proof*⟩

lemma *trace-core-closure-step*:
assumes *trace-core-eq core1 core2 E IA IU p q*
and *trace-core-closure core1 core2 E IA IU p q p′ q′*
shows *trace-core-closure-step-cpoke*:
e ∈ *E* ⟹ *weight-spmf* (*bind-spmf p′* (λ*s*. *cpoke core1 s e*)) = *weight-spmf* (*bind-spmf q′* (λ*s*. *cpoke core2 s e*))
(**is** *PROP ?thesis1*)
and *trace-core-closure-step-cfunc-adv*:
ia ∈ *IA* ⟹ *bind-spmf p′* (λ*s1*. *map-spmf fst* (*cfunc-adv core1 s1 ia*)) = *bind-spmf q′* (λ*s2*. *map-spmf fst* (*cfunc-adv core2 s2 ia*))
(**is** *PROP ?thesis2*)
and *trace-core-closure-step-cfunc-usr*:
iu ∈ *IU* ⟹ *bind-spmf p′* (λ*s1*. *map-spmf fst* (*cfunc-usr core1 s1 iu*)) = *bind-spmf q′* (λ*s2*. *map-spmf fst* (*cfunc-usr core2 s2 iu*))
(**is** *PROP ?thesis3*)
⟨*proof*⟩

lemma *trace-core-closure-sim*:
fixes *core1 core2 E IA IU p q*
defines *S* ≡ *trace-core-closure core1 core2 E IA IU p q*
assumes *S p′ q′*
shows *trace-core-closure-sim-cpoke*:
e ∈ *E* ⟹ *S* (*mk-lossless* (*bind-spmf p′* (λ*s*. *cpoke core1 s e*))) (*mk-lossless* (*bind-spmf q′* (λ*s*. *cpoke core2 s e*)))
(**is** *PROP ?thesis1*)
and *trace-core-closure-sim-cfunc-adv*: *ia* ∈ *IA*
⟹ *S* (*cond-spmf-fst* (*bind-spmf p′* (λ*s1*. *cfunc-adv core1 s1 ia*)) *oa*)
(*cond-spmf-fst* (*bind-spmf q′* (λ*s2*. *cfunc-adv core2 s2 ia*)) *oa*)
(**is** *PROP ?thesis2*)
and *trace-core-closure-sim-cfunc-usr*: *iu* ∈ *IU*
⟹ *S* (*cond-spmf-fst* (*bind-spmf p′* (λ*s1*. *cfunc-usr core1 s1 iu*)) *ou*)
(*cond-spmf-fst* (*bind-spmf q′* (λ*s2*. *cfunc-usr core2 s2 iu*)) *ou*)
(**is** *PROP ?thesis3*)
⟨*proof*⟩

proposition *trace-core-eq-complete*:
assumes *trace-core-eq core1 core2 E IA IU p q*
obtains *S*
where *S p q*
and ⋀*p q e*. ⟦ *S p q*; *e* ∈ *E* ⟧ ⟹
weight-spmf (*bind-spmf p* (λ*s*. *cpoke core1 s e*)) = *weight-spmf* (*bind-spmf q* (λ*s*. *cpoke core2 s e*))
and ⋀*p q e*. ⟦ *S p q*; *e* ∈ *E* ⟧ ⟹
S (*mk-lossless* (*bind-spmf p* (λ*s*. *cpoke core1 s e*))) (*mk-lossless* (*bind-spmf q* (λ*s*. *cpoke core2 s e*)))
and ⋀*p q ia*. ⟦ *S p q*; *ia* ∈ *IA* ⟧ ⟹

bind-spmf p (λ*s1. map-spmf fst* (*cfunc-adv core1 s1 ia*)) = *bind-spmf q* (λ*s2. map-spmf fst* (*cfunc-adv core2 s2 ia*))
and ⋀*p q ia oa.* ⟦ *S p q*; *ia* ∈ *IA* ⟧
⟹ *S* (*cond-spmf-fst* (*bind-spmf p* (λ*s1. cfunc-adv core1 s1 ia*)) *oa*) (*cond-spmf-fst* (*bind-spmf q* (λ*s2. cfunc-adv core2 s2 ia*)) *oa*)
and ⋀*p q iu.* ⟦ *S p q*; *iu* ∈ *IU* ⟧ ⟹
bind-spmf p (λ*s1. map-spmf fst* (*cfunc-usr core1 s1 iu*)) = *bind-spmf q* (λ*s2. map-spmf fst* (*cfunc-usr core2 s2 iu*))
and ⋀*p q iu ou.* ⟦ *S p q*; *iu* ∈ *IU* ⟧
⟹ *S* (*cond-spmf-fst* (*bind-spmf p* (λ*s1. cfunc-usr core1 s1 iu*)) *ou*) (*cond-spmf-fst* (*bind-spmf q* (λ*s2. cfunc-usr core2 s2 iu*)) *ou*)
⟨*proof*⟩

type-synonym (*'event*, *'iadv-rest*, *'iusr-rest*, *'oadv-rest*, *'ousr-rest*) *trace-rest* =
(*'iadv-rest* × *'oadv-rest* × *'event list* + *'iusr-rest* × *'ousr-rest* × *'event list*) *list*
⇒ (*'iadv-rest* ⇒ (*'oadv-rest* × *'event list*) *spmf*)
× (*'iusr-rest* ⇒ (*'ousr-rest* × *'event list*) *spmf*)

context
fixes *rest* :: (*'s-rest*, *'event*, *'iadv-rest*, *'iusr-rest*, *'oadv-rest*, *'ousr-rest*, *'more*) *rest-scheme*
begin

primrec *trace-rest'* :: *'s-rest spmf* ⇒ (*'event*, *'iadv-rest*, *'iusr-rest*, *'oadv-rest*, *'ousr-rest*) *trace-rest* **where**
trace-rest' S [] =
(λ*ia. bind-spmf S* (λ*s. map-spmf fst* (*rfunc-adv rest s ia*)),
λ*iu. bind-spmf S* (λ*s. map-spmf fst* (*rfunc-usr rest s iu*)))
| *trace-rest' S* (*obs* # *tr*) = (*case obs of*
Inl (*ia*, *oa*) ⇒ *trace-rest'* (*cond-spmf-fst* (*bind-spmf S* (λ*s. rfunc-adv rest s ia*)) *oa*) *tr*
| *Inr* (*iu*, *ou*) ⇒ *trace-rest'* (*cond-spmf-fst* (*bind-spmf S* (λ*s. rfunc-usr rest s iu*)) *ou*) *tr*)

end

declare *trace-rest'.simps* [*simp del*]
case-of-simps *trace-rest'-unfold*: *trace-rest'.simps*
simps-of-case *trace-rest'-simps* [*simp*]: *trace-rest'-unfold*

context includes *lifting-syntax* **begin**

lemma *trace-rest'-parametric* [*transfer-rule*]:
(*rel-rest' S* (=) *IA IU* (=) (=) *M* ===> *rel-spmf S* ===>
list-all2 (*rel-sum* (*rel-prod IA* (=)) (*rel-prod IU* (=))) ===>
rel-prod (*IA* ===> (=)) (*IU* ===> (=)))
trace-rest' trace-rest'

⟨*proof*⟩

definition *trace-rest-eq*
:: ('*s-rest*, '*event*, '*iadv-rest*, '*iusr-rest*, '*oadv-rest*, '*ousr-rest*, '*more1*) *rest-scheme*
⇒ ('*s-rest'*, '*event*, '*iadv-rest*, '*iusr-rest*, '*oadv-rest*, '*ousr-rest*, '*more2*) *rest-scheme*
⇒ '*iadv-rest set* ⇒ '*iusr-rest set*
⇒ '*s-rest spmf* ⇒ '*s-rest'* *spmf* ⇒ *bool* **where**
trace-rest-eq rest1 rest2 IA IU p q ⟷
(∀ *tr*. *set tr* ⊆ (*IA* × *UNIV*) <+> (*IU* × *UNIV*) ⟶
rel-prod (*eq-onp* (λ*ia*. *ia* ∈ *IA*) ===> (=)) (*eq-onp* (λ*iu*. *iu* ∈ *IU*) ===> (=))
(*trace-rest'* *rest1 p tr*) (*trace-rest'* *rest2 q tr*))

end

lemma *trace-rest-eqD*:
assumes *trace-rest-eq rest1 rest2 IA IU p q*
and *set tr* ⊆ (*IA* × *UNIV*) <+> (*IU* × *UNIV*)
shows *trace-rest-eqD-rfunc-adv*:
ia ∈ *IA* ⟹ *fst* (*trace-rest'* *rest1 p tr*) *ia* = *fst* (*trace-rest'* *rest2 q tr*) *ia*
and *trace-rest-eqD-rfunc-usr*:
iu ∈ *IU* ⟹ *snd* (*trace-rest'* *rest1 p tr*) *iu* = *snd* (*trace-rest'* *rest2 q tr*) *iu*
⟨*proof*⟩

lemma *trace-rest-eqI*:
assumes ⋀*tr ia*. ⟦ *set tr* ⊆ (*IA* × *UNIV*) <+> (*IU* × *UNIV*); *ia* ∈ *IA* ⟧
⟹ *fst* (*trace-rest'* *rest1 p tr*) *ia* = *fst* (*trace-rest'* *rest2 q tr*) *ia*
and ⋀*tr iu*. ⟦ *set tr* ⊆ (*IA* × *UNIV*) <+> (*IU* × *UNIV*); *iu* ∈ *IU* ⟧
⟹ *snd* (*trace-rest'* *rest1 p tr*) *iu* = *snd* (*trace-rest'* *rest2 q tr*) *iu*
shows *trace-rest-eq rest1 rest2 IA IU p q*
⟨*proof*⟩

lemma *trace-rest-return-pmf-None* [*simp*]:
trace-rest' *rest* (*return-pmf None*) *tr* = (λ-. *return-pmf None*, λ-. *return-pmf None*)
⟨*proof*⟩

lemma *rel-rest'-into-trace-rest-eq*: *trace-rest-eq rest rest' IA IU p q*
if *rel-rest'* *S* (=) (*eq-onp* (λ*ia*. *ia* ∈ *IA*)) (*eq-onp* (λ*iu*. *iu* ∈ *IU*)) (=) (=) *M rest rest'*
rel-spmf S p q
⟨*proof*⟩

lemma *trace-rest-eq-simI*:
fixes *rest1* :: ('*s-rest*, '*event*, '*iadv-rest*, '*iusr-rest*, '*oadv-rest*, '*ousr-rest*, '*more*) *rest-scheme*
and *rest2* :: ('*s-rest'*, '*event*, '*iadv-rest*, '*iusr-rest*, '*oadv-rest*, '*ousr-rest*, '*more*) *rest-scheme*
and *S* :: '*s-rest spmf* ⇒ '*s-rest'* *spmf* ⇒ *bool*
assumes *start*: *S p q*

and *step-rfunc-adv*: $\bigwedge p\ q\ ia.\ [\![\ S\ p\ q;\ ia \in IA\ ]\!] \Longrightarrow$
bind-spmf p (*λs1. map-spmf fst* (*rfunc-adv rest1 s1 ia*)) = *bind-spmf q* (*λs2. map-spmf fst* (*rfunc-adv rest2 s2 ia*))
and *sim-rfunc-adv*: $\bigwedge p\ q\ ia\ s1\ s2\ s1'\ s2'\ oa.\ [\![\ S\ p\ q;\ ia \in IA;$
s1 ∈ *set-spmf p*; *s2* ∈ *set-spmf q*; (*oa*, *s1′*) ∈ *set-spmf* (*rfunc-adv rest1 s1 ia*); (*oa*, *s2′*) ∈ *set-spmf* (*rfunc-adv rest2 s2 ia*)]]
⟹ *S* (*cond-spmf-fst* (*bind-spmf p* (*λs1. rfunc-adv rest1 s1 ia*)) *oa*) (*cond-spmf-fst* (*bind-spmf q* (*λs2. rfunc-adv rest2 s2 ia*)) *oa*)
and *step-rfunc-usr*: $\bigwedge p\ q\ iu.\ [\![\ S\ p\ q;\ iu \in IU\ ]\!] \Longrightarrow$
bind-spmf p (*λs1. map-spmf fst* (*rfunc-usr rest1 s1 iu*)) = *bind-spmf q* (*λs2. map-spmf fst* (*rfunc-usr rest2 s2 iu*))
and *sim-rfunc-usr*: $\bigwedge p\ q\ iu\ s1\ s2\ s1'\ s2'\ ou.\ [\![\ S\ p\ q;\ iu \in IU;$
s1 ∈ *set-spmf p*; *s2* ∈ *set-spmf q*; (*ou*, *s1′*) ∈ *set-spmf* (*rfunc-usr rest1 s1 iu*); (*ou*, *s2′*) ∈ *set-spmf* (*rfunc-usr rest2 s2 iu*)]]
⟹ *S* (*cond-spmf-fst* (*bind-spmf p* (*λs1. rfunc-usr rest1 s1 iu*)) *ou*) (*cond-spmf-fst* (*bind-spmf q* (*λs2. rfunc-usr rest2 s2 iu*)) *ou*)
shows *trace-rest-eq rest1 rest2 IA IU p q*
⟨*proof*⟩

context
fixes *rest* :: (*'s-rest*, *'event*, *'iadv-rest*, *'iusr-rest*, *'oadv-rest*, *'ousr-rest*, *'more*) *rest-scheme*
begin

fun *trace-rest-aux*
:: *'s-rest spmf* ⇒ (*'iadv-rest* × *'oadv-rest* × *'event list* + *'iusr-rest* × *'ousr-rest* × *'event list*) *list* ⇒ *'s-rest spmf* **where**
trace-rest-aux p [] = *p*
| *trace-rest-aux p* (*Inl* (*ia*, *oaes*) # *tr*) = *trace-rest-aux* (*cond-spmf-fst* (*bind-spmf p* (*λs. rfunc-adv rest s ia*)) *oaes*) *tr*
| *trace-rest-aux p* (*Inr* (*iu*, *oues*) # *tr*) = *trace-rest-aux* (*cond-spmf-fst* (*bind-spmf p* (*λs. rfunc-usr rest s iu*)) *oues*) *tr*

end

lemma *trace-rest-conv-trace-rest-aux*:
trace-rest′ rest p tr =
(*λia. bind-spmf* (*trace-rest-aux rest p tr*) (*λs. map-spmf fst* (*rfunc-adv rest s ia*)),
λiu. bind-spmf (*trace-rest-aux rest p tr*) (*λs. map-spmf fst* (*rfunc-usr rest s iu*)))
⟨*proof*⟩

lemma *trace-rest-aux-append*:
trace-rest-aux rest p (*tr* @ *tr′*) = *trace-rest-aux rest* (*trace-rest-aux rest p tr*) *tr′*
⟨*proof*⟩

inductive *trace-rest-closure*
:: (*'s-rest*, *'event*, *'iadv-rest*, *'iusr-rest*, *'oadv-rest*, *'ousr-rest*, *'more*) *rest-scheme*
⇒ (*'s-rest′*, *'event*, *'iadv-rest*, *'iusr-rest*, *'oadv-rest*, *'ousr-rest*, *'more′*) *rest-scheme*
⇒ *'iadv-rest set* ⇒ *'iusr-rest set*

$\Rightarrow$ *'s-rest spmf* $\Rightarrow$ *'s-rest' spmf* $\Rightarrow$ *'s-rest spmf* $\Rightarrow$ *'s-rest' spmf* $\Rightarrow$ *bool*
for *rest1 rest2 IA IU p q* **where**
trace-rest-closure rest1 rest2 IA IU p q (*trace-rest-aux rest1 p tr*) (*trace-rest-aux rest2 q tr*)
if *set tr* $\subseteq$ *IA* $\times$ *UNIV* <+> *IU* $\times$ *UNIV*

lemma *trace-rest-closure-start*: *trace-rest-closure rest1 rest2 IA IU p q p q*
⟨*proof*⟩

lemma *trace-rest-closure-step*:
assumes *trace-rest-eq rest1 rest2 IA IU p q*
and *trace-rest-closure rest1 rest2 IA IU p q p' q'*
shows *trace-rest-closure-step-rfunc-adv*:
ia $\in$ *IA* $\Longrightarrow$ *bind-spmf p'* (*λs1. map-spmf fst* (*rfunc-adv rest1 s1 ia*)) = *bind-spmf q'* (*λs2. map-spmf fst* (*rfunc-adv rest2 s2 ia*))
(**is** *PROP ?thesis1*)
and *trace-rest-closure-step-rfunc-usr*:
iu $\in$ *IU* $\Longrightarrow$ *bind-spmf p'* (*λs1. map-spmf fst* (*rfunc-usr rest1 s1 iu*)) = *bind-spmf q'* (*λs2. map-spmf fst* (*rfunc-usr rest2 s2 iu*))
(**is** *PROP ?thesis2*)
⟨*proof*⟩

lemma *trace-rest-closure-sim*:
fixes *rest1 rest2 IA IU p q*
defines *S* $\equiv$ *trace-rest-closure rest1 rest2 IA IU p q*
assumes *S p' q'*
shows *trace-rest-closure-sim-rfunc-adv*: *ia* $\in$ *IA*
$\Longrightarrow$ *S* (*cond-spmf-fst* (*bind-spmf p'* (*λs1. rfunc-adv rest1 s1 ia*)) *oa*)
(*cond-spmf-fst* (*bind-spmf q'* (*λs2. rfunc-adv rest2 s2 ia*)) *oa*)
(**is** *PROP ?thesis1*)
and *trace-rest-closure-sim-rfunc-usr*: *iu* $\in$ *IU*
$\Longrightarrow$ *S* (*cond-spmf-fst* (*bind-spmf p'* (*λs1. rfunc-usr rest1 s1 iu*)) *ou*)
(*cond-spmf-fst* (*bind-spmf q'* (*λs2. rfunc-usr rest2 s2 iu*)) *ou*)
(**is** *PROP ?thesis2*)
⟨*proof*⟩

proposition *trace-rest-eq-complete*:
assumes *trace-rest-eq rest1 rest2 IA IU p q*
obtains *S*
where *S p q*
and $\bigwedge$*p q ia.* ⟦ *S p q*; *ia* $\in$ *IA* ⟧ $\Longrightarrow$
bind-spmf p (*λs1. map-spmf fst* (*rfunc-adv rest1 s1 ia*)) = *bind-spmf q* (*λs2. map-spmf fst* (*rfunc-adv rest2 s2 ia*))
and $\bigwedge$*p q ia oa.* ⟦ *S p q*; *ia* $\in$ *IA* ⟧
$\Longrightarrow$ *S* (*cond-spmf-fst* (*bind-spmf p* (*λs1. rfunc-adv rest1 s1 ia*)) *oa*) (*cond-spmf-fst* (*bind-spmf q* (*λs2. rfunc-adv rest2 s2 ia*)) *oa*)
and $\bigwedge$*p q iu.* ⟦ *S p q*; *iu* $\in$ *IU* ⟧ $\Longrightarrow$
bind-spmf p (*λs1. map-spmf fst* (*rfunc-usr rest1 s1 iu*)) = *bind-spmf q* (*λs2. map-spmf fst* (*rfunc-usr rest2 s2 iu*))

and $\bigwedge p\ q\ iu\ ou.\ [\![\ S\ p\ q;\ iu \in IU\ ]\!]$
$\Longrightarrow S\ (\mathit{cond\text{-}spmf\text{-}fst}\ (\mathit{bind\text{-}spmf}\ p\ (\lambda s1.\ \mathit{rfunc\text{-}usr}\ \mathit{rest1}\ \mathit{s1}\ \mathit{iu}))\ \mathit{ou})\ (\mathit{cond\text{-}spmf\text{-}fst}\ (\mathit{bind\text{-}spmf}\ q\ (\lambda s2.\ \mathit{rfunc\text{-}usr}\ \mathit{rest2}\ \mathit{s2}\ \mathit{iu}))\ \mathit{ou})$
⟨*proof*⟩

definition *callee-of-core*
:: ('*s-core*, '*event*, '*iadv-core*, '*iusr-core*, '*oadv-core*, '*ousr-core*) *core*
⇒ ('*s-core*, '*event* + '*iadv-core* + '*iusr-core*, *unit* + '*oadv-core* + '*ousr-core*) *oracle′* **where**
callee-of-core core =
map-fun id (*map-fun id* (*map-spmf* (*Pair* ()))) (*cpoke core*) $\oplus_O$ *cfunc-adv core* $\oplus_O$ *cfunc-usr core*

lemma *callee-of-core-simps* [*simp*]:
callee-of-core core s (*Inl e*) = *map-spmf* (*Pair* (*Inl* ())) (*cpoke core s e*)
callee-of-core core s (*Inr* (*Inl iadv-core*)) = *map-spmf* (*apfst* (*Inr* ∘ *Inl*)) (*cfunc-adv core s iadv-core*)
callee-of-core core s (*Inr* (*Inr iusr-core*)) = *map-spmf* (*apfst* (*Inr* ∘ *Inr*)) (*cfunc-usr core s iusr-core*)
⟨*proof*⟩

lemma *WT-callee-of-core* [*WT-intro*]:
assumes *WT*: *WT-core* $\mathcal{I}$*-adv* $\mathcal{I}$*-usr I core*
and *I*: *I s*
shows $\mathcal{I}$*-full* $\oplus_{\mathcal{I}}$ ($\mathcal{I}$*-adv* $\oplus_{\mathcal{I}}$ $\mathcal{I}$*-usr*) ⊢*c callee-of-core core s* √
⟨*proof*⟩

lemma *WT-core-callee-invariant-on* [*WT-intro*]:
assumes *WT*: *WT-core* $\mathcal{I}$*-adv* $\mathcal{I}$*-usr I core*
shows *callee-invariant-on* (*callee-of-core core*) *I* ($\mathcal{I}$*-full* $\oplus_{\mathcal{I}}$ ($\mathcal{I}$*-adv* $\oplus_{\mathcal{I}}$ $\mathcal{I}$*-usr*))
⟨*proof*⟩

definition *callee-of-rest*
:: ('*s-rest*, '*event*, '*iadv-rest*, '*iusr-rest*, '*oadv-rest*, '*ousr-rest*, '*more*) *rest-scheme*
⇒ ('*s-rest*, '*iadv-rest* + '*iusr-rest*, '*oadv-rest* × '*event list* + '*ousr-rest* × '*event list*) *oracle′* **where**
callee-of-rest rest = *rfunc-adv rest* $\oplus_O$ *rfunc-usr rest*

lemma *callee-of-rest-simps* [*simp*]:
callee-of-rest rest s (*Inl iadv-rest*) = *map-spmf* (*apfst Inl*) (*rfunc-adv rest s iadv-rest*)
callee-of-rest rest s (*Inr iusr-rest*) = *map-spmf* (*apfst Inr*) (*rfunc-usr rest s iusr-rest*)
⟨*proof*⟩

lemma *WT-callee-of-rest* [*WT-intro*]:
assumes *WT*: *WT-rest* $\mathcal{I}$*-adv* $\mathcal{I}$*-usr I rest*
and *I*: *I s*
shows *e*$\mathcal{I}$ $\mathcal{I}$*-adv* $\oplus_{\mathcal{I}}$ *e*$\mathcal{I}$ $\mathcal{I}$*-usr* ⊢*c callee-of-rest rest s* √

⟨*proof*⟩

fun *fuse-callee*
:: ('*iadv-core* + '*iadv-rest*) + ('*iusr-core* + '*iusr-rest*) ⇒
((′*oadv-core* + ′*oadv-rest*) + (′*ousr-core* + ′*ousr-rest*),
(′*event* + ′*iadv-core* + ′*iusr-core*) + (′*iadv-rest* + ′*iusr-rest*),
(*unit* + ′*oadv-core* + ′*ousr-core*) + (′*oadv-rest* × ′*event list* + ′*ousr-rest* × ′*event list*)) *gpv*
where
fuse-callee (*Inl* (*Inl iadv-core*)) = *Pause* (*Inl* (*Inr* (*Inl iadv-core*))) (λ*x*. *case x of*
Inl (*Inr* (*Inl oadv-core*)) ⇒ *Done* (*Inl* (*Inl oadv-core*))
| - ⇒ *Fail*)
| *fuse-callee* (*Inl* (*Inr iadv-rest*)) = *Pause* (*Inr* (*Inl iadv-rest*)) (λ*x*. *case x of*
Inr (*Inl* (*oadv-rest*, *es*)) ⇒ *bind-gpv* (*pauses* (*map* (*Inl* ∘ *Inl*) *es*)) (λ-. *Done* (*Inl* (*Inr oadv-rest*)))
| - ⇒ *Fail*)
| *fuse-callee* (*Inr* (*Inl iusr-core*)) = *Pause* (*Inl* (*Inr* (*Inr iusr-core*))) (λ*x*. *case x of*
Inl (*Inr* (*Inr oadv-core*)) ⇒ *Done* (*Inr* (*Inl oadv-core*)))
| *fuse-callee* (*Inr* (*Inr iusr-rest*)) = *Pause* (*Inr* (*Inr iusr-rest*)) (λ*x*. *case x of*
Inr (*Inr* (*ousr-rest*, *es*)) ⇒ *bind-gpv* (*pauses* (*map* (*Inl* ∘ *Inl*) *es*)) (λ-. *Done* (*Inr* (*Inr ousr-rest*))))

case-of-simps *fuse-callee-case*: *fuse-callee.simps*

definition *fuse-converter*
:: (('*iadv-core* + '*iadv-rest*) + ('*iusr-core* + '*iusr-rest*),
('*oadv-core* + '*oadv-rest*) + ('*ousr-core* + '*ousr-rest*),
('*event* + '*iadv-core* + '*iusr-core*) + ('*iadv-rest* + '*iusr-rest*),
(*unit* + ′*oadv-core* + ′*ousr-core*) + (′*oadv-rest* × ′*event list* + ′*ousr-rest* × ′*event list*)) *converter*
where
fuse-converter = *converter-of-callee* (*stateless-callee fuse-callee*) ()

lemma *fuse-converter*:
resource-of-oracle (*fused-resource.fuse core rest*) (*s-core*, *s-rest*) =
fuse-converter ▷ (*resource-of-oracle* (*callee-of-core core*) *s-core* ∥ *resource-of-oracle* (*callee-of-rest rest*) *s-rest*)
⟨*proof*⟩

lemma *trace-eq-callee-of-coreI*:
trace-callee-eq (*callee-of-core core1*) (*callee-of-core core2*) (*E* <+> *IA* <+> *IU*) *p q*
if *trace-core-eq core1 core2 E IA IU p q*
⟨*proof*⟩

lemma *trace-eq-callee-of-restI*:
trace-callee-eq (*callee-of-rest rest1*) (*callee-of-rest rest2*) (*IA* <+> *IU*) *p q*

if *trace-rest-eq rest1 rest2 IA IU p q*
⟨*proof*⟩

lemma *trace-callee-resource-of-oracle*:
trace-callee run-resource (*map-spmf* (*resource-of-oracle callee*) *p*) = *trace-callee callee p*
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *trace-callee-resource-of-oracle′*:
trace-callee run-resource (*return-spmf* (*resource-of-oracle callee s*)) = *trace-callee callee* (*return-spmf s*)
⟨*proof*⟩

lemma *trace-eq-resource-of-oracle*:
trace-eq A (*map-spmf* (*resource-of-oracle callee1*) *p*) (*map-spmf* (*resource-of-oracle callee2*) *q*) =
trace-callee-eq callee1 callee2 A p q
⟨*proof*⟩

lemma *WT-fuse-converter* [*WT-intro*]:
(*ℐAC* ⊕ℐ *map-ℐ id fst ℐAR*) ⊕ℐ (*ℐUC* ⊕ℐ *map-ℐ id fst ℐUR*), (*ℐE* ⊕ℐ (*ℐAC* ⊕ℐ *ℐUC*)) ⊕ℐ (*ℐAR* ⊕ℐ *ℐUR*) ⊢C *fuse-converter* √
if ∀ *x*. ∀ (*y*, *es*)∈*responses-ℐ ℐAR x*. *set es* ⊆ *outs-ℐ ℐE* ∀ *x*. ∀ (*y*, *es*)∈*responses-ℐ ℐUR x*. *set es* ⊆ *outs-ℐ ℐE*
⟨*proof*⟩

theorem *fuse-trace-eq*:
fixes *core1* :: (*'s-core*, *'event*, *'iadv-core*, *'iusr-core*, *'oadv-core*, *'ousr-core*) *core*
and *core2* :: (*'s-core′*, *'event*, *'iadv-core*, *'iusr-core*, *'oadv-core*, *'ousr-core*) *core*
and *rest1* :: (*'s-rest*, *'event*, *'iadv-rest*, *'iusr-rest*, *'oadv-rest*, *'ousr-rest*, *'more1*) *rest-scheme*
and *rest2* :: (*'s-rest′*, *'event*, *'iadv-rest*, *'iusr-rest*, *'oadv-rest*, *'ousr-rest*, *'more2*) *rest-scheme*
assumes *core*: *trace-core-eq core1 core2* (*outs-ℐ ℐE*) (*outs-ℐ ℐCA*) (*outs-ℐ ℐCU*) (*return-spmf s-core*) (*return-spmf s-core′*)
and *rest*: *trace-rest-eq rest1 rest2* (*outs-ℐ ℐRA*) (*outs-ℐ ℐRU*) (*return-spmf s-rest*) (*return-spmf s-rest′*)
and *IC1*: *callee-invariant-on* (*callee-of-core core1*) *IC1* (*ℐE* ⊕ℐ (*ℐCA* ⊕ℐ *ℐCU*)) *IC1 s-core*
and *IC2*: *callee-invariant-on* (*callee-of-core core2*) *IC2* (*ℐE* ⊕ℐ (*ℐCA* ⊕ℐ *ℐCU*)) *IC2 s-core′*
and *IR1*: *callee-invariant-on* (*callee-of-rest rest1*) *IR1* (*ℐRA* ⊕ℐ *ℐRU*) *IR1 s-rest*
and *IR2*: *callee-invariant-on* (*callee-of-rest rest2*) *IR2* (*ℐRA* ⊕ℐ *ℐRU*) *IR2 s-rest′*
and *E1* [*WT-intro*]: ∀ *x*. ∀ (*y*, *es*)∈*responses-ℐ ℐRA x*. *set es* ⊆ *outs-ℐ ℐE*
and *E2* [*WT-intro*]: ∀ *x*. ∀ (*y*, *es*)∈*responses-ℐ ℐRU x*. *set es* ⊆ *outs-ℐ ℐE*
shows *trace-callee-eq* (*fused-resource.fuse core1 rest1*) (*fused-resource.fuse core2*

rest2)
((*outs-I* $\mathcal{I}CA$ <+> *outs-I* $\mathcal{I}RA$) <+> (*outs-I* $\mathcal{I}CU$ <+> *outs-I* $\mathcal{I}RU$))
(*return-spmf* (*s-core*, *s-rest*)) (*return-spmf* (*s-core'*, *s-rest'*))
⟨*proof*⟩

inductive *trace-eq-simcl* :: ('*s1 spmf* ⇒ '*s2 spmf* ⇒ *bool*) ⇒ '*s1 spmf* ⇒ '*s2 spmf* ⇒ *bool*
for *S* **where**
base: *trace-eq-simcl S p q* **if** *S p q* **for** *p q*
| *bind-nat*: *trace-eq-simcl S* (*bind-spmf p f*) (*bind-spmf p g*)
if ⋀*x* :: *nat*. *x* ∈ *set-spmf p* ⟹ *S* (*f x*) (*g x*)

lemma *trace-eq-simcl-bindI* [*intro?*]: *trace-eq-simcl S* (*bind-spmf p f*) (*bind-spmf p g*)
if ⋀*x*. *x* ∈ *set-spmf p* ⟹ *S* (*f x*) (*g x*)
⟨*proof*⟩

lemma *trace-eq-simcl-bind*: *trace-eq-simcl S* (*bind-spmf p f*) (*bind-spmf p g*)
if ∗: ⋀*x* :: '*a*. *x* ∈ *set-spmf p* ⟹ *trace-eq-simcl S* (*f x*) (*g x*)
⟨*proof*⟩

lemma *trace-eq-simcl-bind1-scale*: *trace-eq-simcl S* (*bind-spmf p f*) (*scale-spmf* (*weight-spmf p*) *q*)
if ∀ *x*∈*set-spmf p*. *trace-eq-simcl S* (*f x*) *q*
⟨*proof*⟩

lemma *trace-eq-simcl-bind1*: *trace-eq-simcl S* (*bind-spmf p f*) *q*
if ∀ *x*∈*set-spmf p*. *trace-eq-simcl S* (*f x*) *q lossless-spmf p*
⟨*proof*⟩

lemma *trace-eq-simcl-bind2-scale*: *trace-eq-simcl S* (*scale-spmf* (*weight-spmf q*) *p*) (*bind-spmf q f*)
if ∀ *x*∈*set-spmf q*. *trace-eq-simcl S p* (*f x*)
⟨*proof*⟩

lemma *trace-eq-simcl-bind2*: *trace-eq-simcl S p* (*bind-spmf q f*)
if ∀ *x*∈*set-spmf q*. *trace-eq-simcl S p* (*f x*) *lossless-spmf q*
⟨*proof*⟩

lemma *trace-eq-simcl-return-pmf-None* [*simp*, *intro*!]: *trace-eq-simcl S* (*return-pmf None*) (*return-pmf None*)
for *S* :: '*s1 spmf* ⇒ '*s2 spmf* ⇒ *bool*
⟨*proof*⟩

lemma *trace-eq-simcl-map*: *trace-eq-simcl S* (*map-spmf f p*) (*map-spmf g p*)
if ∀ *x*∈*set-spmf p*. *S* (*return-spmf* (*f x*)) (*return-spmf* (*g x*))
⟨*proof*⟩

lemma *trace-eq-simcl-map1*: *trace-eq-simcl S* (*map-spmf f p*) *q*
if ∀ *x*∈*set-spmf p*. *trace-eq-simcl S* (*return-spmf* (*f x*)) *q lossless-spmf p*
⟨*proof*⟩

lemma *trace-eq-simcl-map2*: *trace-eq-simcl S p* (*map-spmf f q*)
if ∀ *x*∈*set-spmf q*. *trace-eq-simcl S p* (*return-spmf* (*f x*)) *lossless-spmf q*
⟨*proof*⟩

lemma *trace-eq-simcl-return-spmf* [*simp*]: *trace-eq-simcl S* (*return-spmf x*) (*return-spmf y*) ⟷ *S* (*return-spmf x*) (*return-spmf y*)
⟨*proof*⟩

lemma *trace-eq-simcl-callee*:
fixes *callee1* :: ('*a*, '*b*, '*s1*) *callee* **and** *callee2* :: ('*a*, '*b*, '*s2*) *callee*
assumes *step*: ⋀*p q a*. ⟦ *S p q*; *a* ∈ *A* ⟧ ⟹
bind-spmf p (λ*s*. *map-spmf fst* (*callee1 s a*)) = *bind-spmf q* (λ*s*. *map-spmf fst* (*callee2 s a*))
and *sim*: ⋀*p q a res b s'*. ⟦ *S p q*; *a* ∈ *A*; *res* ∈ *set-spmf q*; (*b*, *s'*) ∈ *set-spmf* (*callee2 res a*) ⟧
⟹ *trace-eq-simcl S* (*cond-spmf-fst* (*bind-spmf p* (λ*s*. *callee1 s a*)) *b*)
(*cond-spmf-fst* (*bind-spmf q* (λ*s*. *callee2 s a*)) *b*)
and *start*: *trace-eq-simcl S p q* **and** *a*: *a* ∈ *A*
shows *trace-eq-simcl-callee-step*: *bind-spmf p* (λ*s*. *map-spmf fst* (*callee1 s a*)) = *bind-spmf q* (λ*s*. *map-spmf fst* (*callee2 s a*)) (**is** *?step*)
and *trace-eq-simcl-callee-sim*: ⋀*res b s'*. ⟦ *res* ∈ *set-spmf q*; (*b*, *s'*) ∈ *set-spmf* (*callee2 res a*) ⟧
⟹ *trace-eq-simcl S* (*cond-spmf-fst* (*bind-spmf p* (λ*s*. *callee1 s a*)) *b*)
(*cond-spmf-fst* (*bind-spmf q* (λ*s*. *callee2 s a*)) *b*) (**is** ⋀*res b s'*. ⟦ *?res res*; *?b res b s'* ⟧ ⟹ *?sim res b s'*)
⟨*proof*⟩

proposition *trace'-eqI-sim-upto*:
fixes *callee1* :: ('*a*, '*b*, '*s1*) *callee* **and** *callee2* :: ('*a*, '*b*, '*s2*) *callee*
assumes *start*: *S p q*
and *step*: ⋀*p q a*. ⟦ *S p q*; *a* ∈ *A* ⟧ ⟹
bind-spmf p (λ*s*. *map-spmf fst* (*callee1 s a*)) = *bind-spmf q* (λ*s*. *map-spmf fst* (*callee2 s a*))
and *sim*: ⋀*p q a res b s'*. ⟦ *S p q*; *a* ∈ *A*; *res* ∈ *set-spmf q*; (*b*, *s'*) ∈ *set-spmf* (*callee2 res a*) ⟧
⟹ *trace-eq-simcl S* (*cond-spmf-fst* (*bind-spmf p* (λ*s*. *callee1 s a*)) *b*)
(*cond-spmf-fst* (*bind-spmf q* (λ*s*. *callee2 s a*)) *b*)
shows *trace-callee-eq callee1 callee2 A p q*
⟨*proof*⟩

lemma *trace-core-eq-simI-upto*:
fixes *core1* :: ('*s-core*, '*event*, '*iadv-core*, '*iusr-core*, '*oadv-core*, '*ousr-core*) *core*
and *core2* :: ('*s-core'*, '*event*, '*iadv-core*, '*iusr-core*, '*oadv-core*, '*ousr-core*) *core*
and *S* :: '*s-core spmf* ⇒ '*s-core'* *spmf* ⇒ *bool*
assumes *start*: *S p q*

and *step-cpoke*: $\bigwedge p\ q\ e.\ [\![\ S\ p\ q;\ e \in E\ ]\!] \Longrightarrow$
weight-spmf (*bind-spmf p* (λ*s*. *cpoke core1 s e*)) = *weight-spmf* (*bind-spmf q* (λ*s*. *cpoke core2 s e*))
and *sim-cpoke*: $\bigwedge p\ q\ e.\ [\![\ S\ p\ q;\ e \in E\ ]\!] \Longrightarrow$
trace-eq-simcl S (*mk-lossless* (*bind-spmf p* (λ*s*. *cpoke core1 s e*))) (*mk-lossless* (*bind-spmf q* (λ*s*. *cpoke core2 s e*)))
and *step-cfunc-adv*: $\bigwedge p\ q\ ia.\ [\![\ S\ p\ q;\ ia \in IA\ ]\!] \Longrightarrow$
bind-spmf p (λ*s1*. *map-spmf fst* (*cfunc-adv core1 s1 ia*)) = *bind-spmf q* (λ*s2*. *map-spmf fst* (*cfunc-adv core2 s2 ia*))
and *sim-cfunc-adv*: $\bigwedge p\ q\ ia\ s1\ s2\ s1'\ s2'\ oa.\ [\![\ S\ p\ q;\ ia \in IA;$
s1 ∈ *set-spmf p*; *s2* ∈ *set-spmf q*; (*oa*, *s1′*) ∈ *set-spmf* (*cfunc-adv core1 s1 ia*); (*oa*, *s2′*) ∈ *set-spmf* (*cfunc-adv core2 s2 ia*)]]
⟹ *trace-eq-simcl S* (*cond-spmf-fst* (*bind-spmf p* (λ*s1*. *cfunc-adv core1 s1 ia*)) *oa*) (*cond-spmf-fst* (*bind-spmf q* (λ*s2*. *cfunc-adv core2 s2 ia*)) *oa*)
and *step-cfunc-usr*: $\bigwedge p\ q\ iu.\ [\![\ S\ p\ q;\ iu \in IU\ ]\!] \Longrightarrow$
bind-spmf p (λ*s1*. *map-spmf fst* (*cfunc-usr core1 s1 iu*)) = *bind-spmf q* (λ*s2*. *map-spmf fst* (*cfunc-usr core2 s2 iu*))
and *sim-cfunc-usr*: $\bigwedge p\ q\ iu\ s1\ s2\ s1'\ s2'\ ou.\ [\![\ S\ p\ q;\ iu \in IU;$
s1 ∈ *set-spmf p*; *s2* ∈ *set-spmf q*; (*ou*, *s1′*) ∈ *set-spmf* (*cfunc-usr core1 s1 iu*); (*ou*, *s2′*) ∈ *set-spmf* (*cfunc-usr core2 s2 iu*)]]
⟹ *trace-eq-simcl S* (*cond-spmf-fst* (*bind-spmf p* (λ*s1*. *cfunc-usr core1 s1 iu*)) *ou*) (*cond-spmf-fst* (*bind-spmf q* (λ*s2*. *cfunc-usr core2 s2 iu*)) *ou*)
shows *trace-core-eq core1 core2 E IA IU p q*
⟨*proof*⟩

context
fixes *core* :: (*'s-core*, *'event1* + *'event2*, *'iadv-core*, *'iusr-core*, *'oadv-core*, *'ousr-core*) *core*
and *rest* :: (*'s-rest*, *'event2*, *'iadv-rest*, *'iusr-rest*, *'oadv-rest*, *'ousr-rest*, *'more*) *rest-scheme*
begin

primcorec *core-with-rest* ::
(*'s-core* × *'s-rest*, *'event1*, *'iadv-core* + *'iadv-rest*, *'iusr-core* + *'iusr-rest*, *'oadv-core* + *'oadv-rest*, *'ousr-core* + *'ousr-rest*) *core*
where
cpoke core-with-rest = (λ(*s-core*, *s-rest*) *e*. *map-spmf* (λ*s-core′*. (*s-core′*, *s-rest*)) (*cpoke core s-core* (*Inl e*)))
| *cfunc-adv core-with-rest* = (λ(*s-core*, *s-rest*) *iadv*. *case iadv of*
Inl iadv-core ⇒ *map-spmf* (λ(*oadv-core*, *s-core′*). (*Inl oadv-core*, (*s-core′*, *s-rest*))) (*cfunc-adv core s-core iadv-core*)
| *Inr iadv-rest* ⇒
bind-spmf (*rfunc-adv rest s-rest iadv-rest*) (λ((*oadv-rest*, *es*), *s-rest′*).
map-spmf (λ*s-core′*. (*Inr oadv-rest*, (*s-core′*, *s-rest′*))) (*foldl-spmf* (*cpoke core*) (*return-spmf s-core*) (*map Inr es*))))
| *cfunc-usr core-with-rest* = (λ(*s-core*, *s-rest*) *iusr*. *case iusr of*
Inl iusr-core ⇒ *map-spmf* (λ(*ousr-core*, *s-core′*). (*Inl ousr-core*, (*s-core′*,

s-rest))) (*cfunc-usr core s-core iusr-core*)
| *Inr iusr-rest* ⇒
bind-spmf (*rfunc-usr rest s-rest iusr-rest*) (λ((*ousr-rest*, *es*), *s-rest'*).
map-spmf (λ*s-core'*. (*Inr ousr-rest*, (*s-core'*, *s-rest'*))) (*foldl-spmf* (*cpoke core*) (*return-spmf s-core*) (*map Inr es*))))

end

lemma *fuse-core-with-rest*:
fixes *core* :: ('*s-core*, '*event1* + '*event2*, '*iadv-core*, '*iusr-core*, '*oadv-core*, '*ousr-core*) *core*
and *rest1* :: ('*s-rest1*, '*event1*, '*iadv-rest1*, '*iusr-rest1*, '*oadv-rest1*, '*ousr-rest1*, '*more1*) *rest-scheme*
and *rest2* :: ('*s-rest2*, '*event2*, '*iadv-rest2*, '*iusr-rest2*, '*oadv-rest2*, '*ousr-rest2*, '*more2*) *rest-scheme*
shows
fused-resource.fuse core (*parallel-rest rest1 rest2*) (*s-core*, (*s-rest1*, *s-rest2*)) =
map-fun (*map-sum* (*lsumr* ∘ *map-sum id swap-sum*) (*lsumr* ∘ *map-sum id swap-sum*)) (*map-spmf* (*map-prod* (*map-sum* (*map-sum id swap-sum* ∘ *rsuml*) (*map-sum id swap-sum* ∘ *rsuml*)) (*map-prod id prod.swap* ∘ *rprodl*)))
(*fused-resource.fuse* (*core-with-rest core rest2*) *rest1* ((*s-core*, *s-rest2*), *s-rest1*))
⟨*proof*⟩

end
theory *State-Isomorphism*
imports
More-CC
begin

6 State Isomorphism

type-synonym
('*a*, '*b*) *state-iso* = ('*a* ⇒ '*b*) × ('*b* ⇒ '*a*)

definition
state-iso :: ('*a*, '*b*) *state-iso* ⇒ *bool*
where
state-iso ≡ (λ(*f*, *g*). *type-definition f g UNIV*)

definition
apply-state-iso :: ('*s1*, '*s2*) *state-iso* ⇒ ('*s1*, '*i*, '*o*) *oracle'* ⇒ ('*s2*, '*i*, '*o*) *oracle'*
where
apply-state-iso ≡ (λ(*f*, *g*). *map-fun g* (*map-fun id* (*map-spmf* (*map-prod id f*))))

lemma *apply-state-iso-id*: *apply-state-iso* (*id*, *id*) = *id*
⟨*proof*⟩

lemma *apply-state-iso-compose*: *apply-state-iso si1* (*apply-state-iso si2 oracle*) =
apply-state-iso (*map-prod* (λ*f*. *f o* (*fst si2*)) ((*o*) (*snd si2*)) *si1*) *oracle*

⟨*proof*⟩

lemma *apply-wiring-state-iso-assoc*:
apply-wiring wr (*apply-state-iso si oracle*) = *apply-state-iso si* (*apply-wiring wr oracle*)
⟨*proof*⟩

lemma
resource-of-oracle-state-iso:
assumes *state-iso fg*
shows *resource-of-oracle* (*apply-state-iso fg oracle*) *s* = *resource-of-oracle oracle* (*snd fg s*)
⟨*proof*⟩

6.1 Parallel State Isomorphism

definition
parallel-state-iso :: (('*s-core1* × '*s-core2*) × ('*s-rest1* × '*s-rest2*), ('*s-core1* × '*s-rest1*) × ('*s-core2* × '*s-rest2*)) *state-iso*
where
parallel-state-iso =
(λ((*s11*, *s12*), (*s21*, *s22*)). ((*s11*, *s21*), (*s12*, *s22*)), λ((*s11*, *s21*), (*s12*, *s22*)). ((*s11*, *s12*), (*s21*, *s22*)))

lemma
state-iso-parallel-state-iso [*simp*]: *state-iso parallel-state-iso*
⟨*proof*⟩

6.2 Trisplit State Isomorphism

definition
iso-trisplit
where
iso-trisplit =
(λ(((*s11*, *s12*), *s13*), (*s21*, *s22*), *s23*). (((*s11*, *s21*), *s12*, *s22*), *s13*, *s23*), λ(((*s11*, *s21*), *s12*, *s22*), *s13*, *s23*). (((*s11*, *s12*), *s13*), (*s21*, *s22*), *s23*))

lemma
state-iso-fuse-par [*simp*]: *state-iso iso-trisplit*
⟨*proof*⟩

6.3 Assocl-Swap State Isomorphism

definition
iso-swapar
where
iso-swapar = (λ((*sm*, *s1*), *s2*). (*s1*, *sm*, *s2*), λ(*s1*, *sm*, *s2*). ((*sm*, *s1*), *s2*))

lemma
state-iso-swapar [*simp*]: *state-iso iso-swapar*

⟨*proof*⟩

end
theory *Construction-Utility*
imports
Fused-Resource
State-Isomorphism
begin

— Dummy converters that return a constant value on their external interface

primcorec
ldummy-converter :: ('*a* ⇒ '*b*) ⇒ ('*i-cnv*, '*o-cnv*, '*i-res*, '*o-res*) *converter* ⇒
('*a* + '*i-cnv*, '*b* + '*o-cnv*, '*i-res*, '*o-res*) *converter*
where
run-converter (*ldummy-converter f conv*) = (λ*inp*. *case inp of*
Inl x ⇒ *map-gpv* (*map-prod Inl* (λ-. *ldummy-converter f conv*)) *id* (*Done* (*f x*, ()))
| *Inr x* ⇒ *map-gpv* (*map-prod Inr* (λ*c*. *ldummy-converter f c*)) *id* (*run-converter conv x*))

primcorec
rdummy-converter :: ('*a* ⇒ '*b*) ⇒ ('*i-cnv*, '*o-cnv*, '*i-res*, '*o-res*) *converter* ⇒
('*i-cnv* + '*a*, '*o-cnv* + '*b*, '*i-res*, '*o-res*) *converter*
where
run-converter (*rdummy-converter f conv*) = (λ*inp*. *case inp of*
Inl x ⇒ *map-gpv* (*map-prod Inl* (λ*c*. *rdummy-converter f c*)) *id* (*run-converter conv x*)
| *Inr x* ⇒ *map-gpv* (*map-prod Inr* (λ-. *rdummy-converter f conv*)) *id* (*Done* (*f x*, ())))

lemma *ldummy-converter-of-callee*:
ldummy-converter f (*converter-of-callee callee state*) =
converter-of-callee (λ*s q*. *case-sum* (λ*ql*. *Done* (*Inl* (*f ql*), *s*)) (λ*qr*. *map-gpv* (*map-prod Inr id*) *id* (*callee s qr*)) *q*) *state*
⟨*proof*⟩

lemma *rdummy-converter-of-callee*:
rdummy-converter f (*converter-of-callee callee state*) =
converter-of-callee (λ*s q*. *case-sum* (λ*ql*. *map-gpv* (*map-prod Inl id*) *id* (*callee s ql*)) (λ*qr*. *Done* (*Inr* (*f qr*), *s*)) *q*) *state*
⟨*proof*⟩

context
fixes
cnv1 :: ('*icnv-usr1*, '*ocnv-usr1*, '*iusr1-res1* + '*iusr1-res2*, '*ousr1-res1* + '*ousr1-res2*) *converter* **and**
cnv2 :: ('*icnv-usr2*, '*ocnv-usr2*, '*iusr2-res1* + '*iusr2-res2*, '*ousr2-res1* + '*ousr2-res2*) *converter*

begin

— c1r22: a converter that has 1 interface and sends queries to two resources, where the first and second resources have 2 and 2 interfaces respectively

definition
wiring-c1r22-c1r22 :: (*'icnv-usr1* + *'icnv-usr2*, *'ocnv-usr1* + *'ocnv-usr2*,
(*'iusr1-res1* + *'iusr2-res1*) + *'iusr1-res2* + *'iusr2-res2*,
(*'ousr1-res1* + *'ousr2-res1*) + *'ousr1-res2* + *'ousr2-res2*) *converter*
where
wiring-c1r22-c1r22 ≡ (*cnv1* |= *cnv2*) ⊙ *parallel-wiring*

end

— Special wiring converters used for the parallel composition of Fused resources

definition
fused-wiring ::
(((*'iadv-core1* + *'iadv-core2*) + (*'iadv-rest1* + *'iadv-rest2*)) +
((*'iusr-core1* + *'iusr-core2*) + (*'iusr-rest1* + *'iusr-rest2*)),
((*'oadv-core1* + *'oadv-core2*) + (*'oadv-rest1* + *'oadv-rest2*)) +
((*'ousr-core1* + *'ousr-core2*) + (*'ousr-rest1* + *'ousr-rest2*)),
((*'iadv-core1* + *'iadv-rest1*) + (*'iusr-core1* + *'iusr-rest1*)) +
((*'iadv-core2* + *'iadv-rest2*) + (*'iusr-core2* + *'iusr-rest2*)),
((*'oadv-core1* + *'oadv-rest1*) + (*'ousr-core1* + *'ousr-rest1*)) +
((*'oadv-core2* + *'oadv-rest2*) + (*'ousr-core2* + *'ousr-rest2*))) *converter*
where
fused-wiring ≡ (*parallel-wiring* |= *parallel-wiring*) ⊙ *parallel-wiring*

definition
$fused\text{-}wiring_w$
where
$fused\text{-}wiring_w \equiv (parallel\text{-}wiring_w \mid_w parallel\text{-}wiring_w) \circ_w parallel\text{-}wiring_w$

schematic-goal
wiring-fused-wiring[*wiring-intro*]: *wiring ?I1 ?I2 fused-wiring* $fused\text{-}wiring_w$
⟨*proof*⟩

schematic-goal *WT-fused-wiring* [*WT-intro*]: *?I1*, *?I2* ⊢$_C$ *fused-wiring* √
⟨*proof*⟩

context
fixes
cnv1 :: (*'icnv-usr1*, *'ocnv-usr1*, *'iusr1-core1* + *'iusr1-core2*, *'ousr1-core1* + *'ousr1-core2*) *converter* **and**
cnv2 :: (*'icnv-usr2*, *'ocnv-usr2*, *'iusr2-core1* + *'iusr2-core2*, *'ousr2-core1* + *'ousr2-core2*) *converter* **and**
res1 :: ((*'iadv-core1* + *'iadv-rest1*) + (*'iusr1-core1* + *'iusr2-core1*) + *'iusr-rest1*,

(*'oadv-core1* + *'oadv-rest1*) + (*'ousr1-core1* + *'ousr2-core1*) + *'ousr-rest1*) *resource* **and**
res2 :: ((*'iadv-core2* + *'iadv-rest2*) + (*'iusr1-core2* + *'iusr2-core2*) + *'iusr-rest2*,
(*'oadv-core2* + *'oadv-rest2*) + (*'ousr1-core2* + *'ousr2-core2*) + *'ousr-rest2*) *resource*
begin

— Attachement of two c1f22 ('f' instead of 'r' to indicate Fused Resources) converters to two 2-interface Fused Resources, the results will be a new 2-interface Fused Resource

definition
attach-c1f22-c1f22 :: (((*'iadv-core1* + *'iadv-core2*) + *'iadv-rest1* + *'iadv-rest2*) + (*'icnv-usr1* + *'icnv-usr2*) + *'iusr-rest1* + *'iusr-rest2*,
((*'oadv-core1* + *'oadv-core2*) + *'oadv-rest1* + *'oadv-rest2*) + (*'ocnv-usr1* + *'ocnv-usr2*) + *'ousr-rest1* + *'ousr-rest2*) *resource*
where
attach-c1f22-c1f22 = (((*1*$_C$ |$_=$ *1*$_C$) |$_=$ ((*wiring-c1r22-c1r22 cnv1 cnv2*) |$_=$ *1*$_C$)) ⊙ *fused-wiring*) ▷ (*res1* ∥ *res2*)
end

— Properties of Converters attaching to Fused resources
context
fixes
core1 :: (*'s-core1*, *'e1*, *'iadv-core1*, *'iusr-core1*, *'oadv-core1*, *'ousr-core1*) *core* **and**
core2 :: (*'s-core2*, *'e2*, *'iadv-core2*, *'iusr-core2*, *'oadv-core2*, *'ousr-core2*) *core* **and**
rest1 :: (*'s-rest1*, *'e1*, *'iadv-rest1*, *'iusr-rest1*, *'oadv-rest1*, *'ousr-rest1*, *'m1*) *rest-scheme* **and**
rest2 :: (*'s-rest2*, *'e2*, *'iadv-rest2*, *'iusr-rest2*, *'oadv-rest2*, *'ousr-rest2*, *'m2*) *rest-scheme*
begin

lemma *parallel-oracle-fuse*:
apply-wiring fused-wiring$_w$ (*parallel-oracle* (*fused-resource.fuse core1 rest1*) (*fused-resource.fuse core2 rest2*)) =
apply-state-iso parallel-state-iso (*fused-resource.fuse* (*parallel-core core1 core2*) (*parallel-rest rest1 rest2*))
⟨*proof*⟩
end

lemma *attach-callee-fuse*:
attach-callee ((*cnv-adv-core* ‡$_I$ *cnv-adv-rest*) ‡$_I$ *cnv-usr-core* ‡$_I$ *cnv-usr-rest*) (*fused-resource.fuse core rest*) =
apply-state-iso iso-trisplit (*fused-resource.fuse* (*attach-core cnv-adv-core cnv-usr-core core*) (*attach-rest cnv-adv-rest cnv-usr-rest f-init rest*))
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *attach-parallel-fuse′*:
(*CNV cnv-adv-core s-a-c* $|_=$ *CNV cnv-adv-rest s-a-r*) $|_=$ (*CNV cnv-usr-core s-u-c* $|_=$ *CNV cnv-usr-rest s-u-r*) $\triangleright$
RES (*fused-resource.fuse core rest*) (*s-r-c*, *s-r-r*) =
RES (*fused-resource.fuse* (*attach-core cnv-adv-core cnv-usr-core core*) (*attach-rest cnv-adv-rest cnv-usr-rest f-init rest*)) (((*s-a-c*, *s-u-c*), *s-r-c*), ((*s-a-r*, *s-u-r*), *s-r-r*))
⟨*proof*⟩

context
fixes
einit :: *′s-event* **and**
etran :: (*′s-event*, *′ievent*, *′oevent list*) *oracle′* **and**
rest :: (*′s-rest*, *′ievent*, *′iadv-rest*, *′iusr-rest*, *′oadv-rest*, *′ousr-rest*) *rest-wstate*
and
core :: (*′s-core*, *′oevent*, *′iadv-core*, *′iusr-core*, *′oadv-core*, *′ousr-core*) *core*
begin

primcorec
translate-rest :: (*′s-event* × *′s-rest*, *′oevent*, *′iadv-rest*, *′iusr-rest*, *′oadv-rest*, *′ousr-rest*) *rest-wstate*
where
rinit translate-rest = (*einit*, *rinit rest*)
| *rfunc-adv translate-rest* = *translate-eoracle etran* (*extend-state-oracle* (*rfunc-adv rest*))
| *rfunc-usr translate-rest* = *translate-eoracle etran* (*extend-state-oracle* (*rfunc-usr rest*))

primcorec
translate-core :: (*′s-event* × *′s-core*, *′ievent*, *′iadv-core*, *′iusr-core*, *′oadv-core*, *′ousr-core*) *core*
where
cpoke translate-core = (λ(*s-event*, *s-core*) *event*.
bind-spmf (*etran s-event event*) (λ(*events*, *s-event′*).
map-spmf (λ*s-core′*. (*s-event′*, *s-core′*)) (*foldl-spmf* (*cpoke core*) (*return-spmf s-core*) *events*)))
| *cfunc-adv translate-core* = *extend-state-oracle* (*cfunc-adv core*)
| *cfunc-usr translate-core* = *extend-state-oracle* (*cfunc-usr core*)

lemma *WT-translate-rest* [*WT-intro*]:
assumes *WT-rest* $\mathcal{I}$*-adv* $\mathcal{I}$*-usr I-rest rest*
shows *WT-rest* $\mathcal{I}$*-adv* $\mathcal{I}$*-usr* (*pred-prod* (λ*-*. *True*) *I-rest*) *translate-rest*
⟨*proof*⟩

lemma *fused-resource-move-translate*:
fused-resource.fuse core translate-rest = *apply-state-iso iso-swapar* (*fused-resource.fuse translate-core rest*)

⟨*proof*⟩

end

— Moving interfaces between rest and core

lemma
fuse-ishift-core-to-rest:
assumes *cpoke core'* = (λ*s*. *case-sum* (λ*q*. *fn s q*) (*cpoke core s*))
and *cfunc-adv core* = *cfunc-adv core'*
and *cfunc-usr core* = *cfunc-usr core'* ⊕$_O$ (λ*s i*. *map-spmf* (*Pair* (*h-out i*))
(*fn s i*))
and *rfunc-adv rest'* = (λ*s q*. *map-spmf* (*apfst* (*apsnd* (*map Inr*))) (*rfunc-adv
rest s q*))
and *rfunc-usr rest'* = *plus-eoracle* (λ*s i*. *return-spmf* ((*h-out i*, [*i*]), *s*))
(*rfunc-usr rest*)
shows *fused-resource.fuse core rest* = *apply-wiring* (1$_w$ |$_w$ *lassocr*$_w$) (*fused-resource.fuse
core' rest'*) (**is** *?L* = *?R*)
⟨*proof*⟩

lemma *move-simulator-interface*:
defines *x-ifunc* ≡ (λ*ifunc core* (*se*, *sc*) *q*. *do* {
((*out*, *es*), *se'*) ← *ifunc se q*;
sc' ← *foldl-spmf* (*cpoke core*) (*return-spmf sc*) *es*;
return-spmf (*out*, *se'*, *sc'*) })
assumes *cpoke core'* = *cpoke* (*translate-core etran core*)
and *cfunc-adv core'* = †(*cfunc-adv core*) ⊕$_O$ *x-ifunc ifunc core*
and *cfunc-usr core'* = *cfunc-usr* (*translate-core etran core*)
and *rinit rest* = (*einit*, *rinit rest'*)
and *rfunc-adv rest* = (λ*s q*. *case q of*
Inl ql ⇒ *map-spmf* (*apfst* (*map-prod Inl id*)) ((*ifunc*†) *s ql*)
| *Inr qr* ⇒ *map-spmf* (*apfst* (*map-prod Inr id*)) ((*translate-eoracle etran*
(†(*rfunc-adv rest'*))) *s qr*))
and *rfunc-usr rest* = *translate-eoracle etran* (†(*rfunc-usr rest'*))
shows *fused-resource.fuse core rest* = *apply-wiring* (*rassocl*$_w$ |$_w$ (*id*, *id*))
(*apply-state-iso* (*rprodl o* (*apfst prod.swap*), (*apfst prod.swap*) *o lprodr*)
(*fused-resource.fuse core' rest'*))
(**is** *?L* = *?R*)
⟨*proof*⟩

end
theory *Concrete-Security*
imports
Observe-Failure

Construction-Utility
begin

7 Concrete security definition

locale *constructive-security-aux-obsf* =
  fixes *real-resource* :: ('*a* + '*e*, '*b* + '*f*) *resource*
    and *ideal-resource* :: ('*c* + '*e*, '*d* + '*f*) *resource*
    and *sim* :: ('*a*, '*b*, '*c*, '*d*) *converter*
    and $\mathcal{I}$*-real* :: ('*a*, '*b*) $\mathcal{I}$
    and $\mathcal{I}$*-ideal* :: ('*c*, '*d*) $\mathcal{I}$
    and $\mathcal{I}$*-common* :: ('*e*, '*f*) $\mathcal{I}$
    and *adv* :: *real*
  assumes *WT-real* [*WT-intro*]: $\mathcal{I}$*-real* $\oplus_{\mathcal{I}}$ $\mathcal{I}$*-common* $\vdash_{res}$ *real-resource* $\surd$
    and *WT-ideal* [*WT-intro*]: $\mathcal{I}$*-ideal* $\oplus_{\mathcal{I}}$ $\mathcal{I}$*-common* $\vdash_{res}$ *ideal-resource* $\surd$
    and *WT-sim* [*WT-intro*]: $\mathcal{I}$*-real*, $\mathcal{I}$*-ideal* $\vdash_C$ *sim* $\surd$
    and *pfinite-sim* [*pfinite-intro*]: *pfinite-converter* $\mathcal{I}$*-real* $\mathcal{I}$*-ideal sim*
    and *adv-nonneg*: $0 \leq$ *adv*

locale *constructive-security-sim-obsf* =
  fixes *real-resource* :: ('*a* + '*e*, '*b* + '*f*) *resource*
    and *ideal-resource* :: ('*c* + '*e*, '*d* + '*f*) *resource*
    and *sim* :: ('*a*, '*b*, '*c*, '*d*) *converter*
    and $\mathcal{I}$*-real* :: ('*a*, '*b*) $\mathcal{I}$
    and $\mathcal{I}$*-common* :: ('*e*, '*f*) $\mathcal{I}$
    and $\mathcal{A}$ :: ('*a* + '*e*, '*b* + '*f*) *distinguisher-obsf*
    and *adv* :: *real*
  assumes *adv*: ⟦ *exception-*$\mathcal{I}$ ($\mathcal{I}$*-real* $\oplus_{\mathcal{I}}$ $\mathcal{I}$*-common*) $\vdash_g$ $\mathcal{A}$ $\surd$ ⟧
    ⟹ *advantage* $\mathcal{A}$ (*obsf-resource* (*sim* $|_=$ 1_C ▷ *ideal-resource*)) (*obsf-resource* (*real-resource*)) ≤ *adv*

locale *constructive-security-obsf* = *constructive-security-aux-obsf real-resource ideal-resource sim* $\mathcal{I}$*-real* $\mathcal{I}$*-ideal* $\mathcal{I}$*-common adv*
  + *constructive-security-sim-obsf real-resource ideal-resource sim* $\mathcal{I}$*-real* $\mathcal{I}$*-common* $\mathcal{A}$ *adv*
  for *real-resource* :: ('*a* + '*e*, '*b* + '*f*) *resource*
    and *ideal-resource* :: ('*c* + '*e*, '*d* + '*f*) *resource*
    and *sim* :: ('*a*, '*b*, '*c*, '*d*) *converter*
    and $\mathcal{I}$*-real* :: ('*a*, '*b*) $\mathcal{I}$
    and $\mathcal{I}$*-ideal* :: ('*c*, '*d*) $\mathcal{I}$
    and $\mathcal{I}$*-common* :: ('*e*, '*f*) $\mathcal{I}$
    and $\mathcal{A}$ :: ('*a* + '*e*, '*b* + '*f*) *distinguisher-obsf*
    and *adv* :: *real*
begin

lemma *constructive-security-aux-obsf*: *constructive-security-aux-obsf real-resource ideal-resource sim* $\mathcal{I}$*-real* $\mathcal{I}$*-ideal* $\mathcal{I}$*-common adv* ⟨*proof*⟩
lemma *constructive-security-sim-obsf*: *constructive-security-sim-obsf real-resource ideal-resource sim* $\mathcal{I}$*-real* $\mathcal{I}$*-common* $\mathcal{A}$ *adv* ⟨*proof*⟩

end

context *constructive-security-aux-obsf* **begin**

lemma *constructive-security-obsf-refl*:
constructive-security-obsf real-resource ideal-resource sim $\mathcal{I}$*-real* $\mathcal{I}$*-ideal* $\mathcal{I}$*-common* $\mathcal{A}$
(*advantage* $\mathcal{A}$ (*obsf-resource* (*sim* $\models$ 1_C $\triangleright$ *ideal-resource*)) (*obsf-resource* (*real-resource*)))
⟨*proof*⟩

end

lemma *constructive-security-obsf-absorb-cong*:
assumes *sec*: *constructive-security-obsf real-resource ideal-resource sim* $\mathcal{I}$*-real* $\mathcal{I}$*-ideal* $\mathcal{I}$*-common* (*absorb* $\mathcal{A}$ *cnv*) *adv*
and [*WT-intro*]: *exception-*$\mathcal{I}$ $\mathcal{I}$, *exception-*$\mathcal{I}$ ($\mathcal{I}$*-real* $\oplus_{\mathcal{I}}$ $\mathcal{I}$*-common*) $\vdash_C$ *cnv* $\surd$ *exception-*$\mathcal{I}$ $\mathcal{I}$, *exception-*$\mathcal{I}$ ($\mathcal{I}$*-real* $\oplus_{\mathcal{I}}$ $\mathcal{I}$*-common*) $\vdash_C$ *cnv'* $\surd$ *exception-*$\mathcal{I}$ $\mathcal{I}$ $\vdash g$ $\mathcal{A}$ $\surd$
and *cong*: *exception-*$\mathcal{I}$ $\mathcal{I}$, *exception-*$\mathcal{I}$ ($\mathcal{I}$*-real* $\oplus_{\mathcal{I}}$ $\mathcal{I}$*-common*) $\vdash_C$ *cnv* $\sim$ *cnv'*
shows *constructive-security-obsf real-resource ideal-resource sim* $\mathcal{I}$*-real* $\mathcal{I}$*-ideal* $\mathcal{I}$*-common* (*absorb* $\mathcal{A}$ *cnv'*) *adv*
⟨*proof*⟩

lemma *constructive-security-obsf-sim-cong*:
assumes *sec*: *constructive-security-obsf real-resource ideal-resource sim* $\mathcal{I}$*-real* $\mathcal{I}$*-ideal* $\mathcal{I}$*-common* $\mathcal{A}$ *adv*
and *cong*: $\mathcal{I}$*-real*, $\mathcal{I}$*-ideal* $\vdash_C$ *sim* $\sim$ *sim'*
and *pfinite* [*pfinite-intro*]: *pfinite-converter* $\mathcal{I}$*-real* $\mathcal{I}$*-ideal sim'*
shows *constructive-security-obsf real-resource ideal-resource sim'* $\mathcal{I}$*-real* $\mathcal{I}$*-ideal* $\mathcal{I}$*-common* $\mathcal{A}$ *adv*
⟨*proof*⟩

lemma *constructive-security-obsfI-core-rest* [*locale-witness*]:
assumes *constructive-security-aux-obsf real-resource ideal-resource sim* $\mathcal{I}$*-real* $\mathcal{I}$*-ideal* ($\mathcal{I}$*-common-core* $\oplus_{\mathcal{I}}$ $\mathcal{I}$*-common-rest*) *adv*
and *adv*: ⟦ *exception-*$\mathcal{I}$ ($\mathcal{I}$*-real* $\oplus_{\mathcal{I}}$ ($\mathcal{I}$*-common-core* $\oplus_{\mathcal{I}}$ $\mathcal{I}$*-common-rest*)) $\vdash g$ $\mathcal{A}$ $\surd$ ⟧
$\Longrightarrow$ *advantage* $\mathcal{A}$ (*obsf-resource* (*sim* $\models$ (1_C $\models$ 1_C) $\triangleright$ *ideal-resource*)) (*obsf-resource* (*real-resource*)) $\leq$ *adv*
shows *constructive-security-obsf real-resource ideal-resource sim* $\mathcal{I}$*-real* $\mathcal{I}$*-ideal* ($\mathcal{I}$*-common-core* $\oplus_{\mathcal{I}}$ $\mathcal{I}$*-common-rest*) $\mathcal{A}$ *adv*
⟨*proof*⟩

7.1 Composition theorems

theorem *constructive-security-obsf-composability*:
fixes *real*

assumes *constructive-security-obsf middle ideal sim-inner* $\mathcal{I}$*-middle* $\mathcal{I}$*-inner* $\mathcal{I}$*-common* (*absorb* $\mathcal{A}$ (*obsf-converter* (*sim-outer* $|_=$ 1_C))) *adv1*
assumes *constructive-security-obsf real middle sim-outer* $\mathcal{I}$*-real* $\mathcal{I}$*-middle* $\mathcal{I}$*-common* $\mathcal{A}$ *adv2*
shows *constructive-security-obsf real ideal* (*sim-outer* $\odot$ *sim-inner*) $\mathcal{I}$*-real* $\mathcal{I}$*-inner* $\mathcal{I}$*-common* $\mathcal{A}$ (*adv1* + *adv2*)
⟨*proof*⟩

theorem *constructive-security-obsf-lifting*:
assumes *sec*: *constructive-security-aux-obsf real-resource ideal-resource sim* $\mathcal{I}$*-real* $\mathcal{I}$*-ideal* $\mathcal{I}$*-common adv*
and *sec2*: *exception-*$\mathcal{I}$ ($\mathcal{I}$*-real*$'$ $\oplus_{\mathcal{I}}$ $\mathcal{I}$*-common*$'$) $\vdash g$ $\mathcal{A}$ $\surd$
$\Longrightarrow$ *constructive-security-sim-obsf real-resource ideal-resource sim* $\mathcal{I}$*-real* $\mathcal{I}$*-common* (*absorb* $\mathcal{A}$ (*obsf-converter* (*w-adv-real* $|_=$ *w-usr*))) *adv*
(**is** - $\Longrightarrow$ *constructive-security-sim-obsf - - - - - ?*$\mathcal{A}$ -)
assumes *WT-usr* [*WT-intro*]: $\mathcal{I}$*-common*$'$, $\mathcal{I}$*-common* $\vdash_C$ *w-usr* $\surd$
and *pfinite* [*pfinite-intro*]: *pfinite-converter* $\mathcal{I}$*-common*$'$ $\mathcal{I}$*-common w-usr*
and *WT-adv-real* [*WT-intro*]: $\mathcal{I}$*-real*$'$, $\mathcal{I}$*-real* $\vdash_C$ *w-adv-real* $\surd$
and *WT-w-adv-ideal* [*WT-intro*]: $\mathcal{I}$*-ideal*$'$, $\mathcal{I}$*-ideal* $\vdash_C$ *w-adv-ideal* $\surd$
and *WT-adv-ideal-inv* [*WT-intro*]: $\mathcal{I}$*-ideal*, $\mathcal{I}$*-ideal*$'$ $\vdash_C$ *w-adv-ideal-inv* $\surd$
and *ideal-inverse*: $\mathcal{I}$*-ideal*, $\mathcal{I}$*-ideal* $\vdash_C$ *w-adv-ideal-inv* $\odot$ *w-adv-ideal* $\sim$ 1_C
and *pfinite-real* [*pfinite-intro*]: *pfinite-converter* $\mathcal{I}$*-real*$'$ $\mathcal{I}$*-real w-adv-real*
and *pfinite-ideal* [*pfinite-intro*]: *pfinite-converter* $\mathcal{I}$*-ideal* $\mathcal{I}$*-ideal*$'$ *w-adv-ideal-inv*
shows *constructive-security-obsf* (*w-adv-real* $|_=$ *w-usr* $\triangleright$ *real-resource*) (*w-adv-ideal* $|_=$ *w-usr* $\triangleright$ *ideal-resource*) (*w-adv-real* $\odot$ *sim* $\odot$ *w-adv-ideal-inv*) $\mathcal{I}$*-real*$'$ $\mathcal{I}$*-ideal*$'$ $\mathcal{I}$*-common*$'$ $\mathcal{A}$ *adv*
(**is** *constructive-security-obsf ?real ?ideal ?sim ?*$\mathcal{I}$*-real ?*$\mathcal{I}$*-ideal* - - -)
⟨*proof*⟩

corollary *constructive-security-obsf-lifting-*:
assumes *sec*: *constructive-security-obsf real-resource ideal-resource sim* $\mathcal{I}$*-real* $\mathcal{I}$*-ideal* $\mathcal{I}$*-common* (*absorb* $\mathcal{A}$ (*obsf-converter* (*w-adv-real* $|_=$ *w-usr*))) *adv*
assumes *WT-usr* [*WT-intro*]: $\mathcal{I}$*-common*$'$, $\mathcal{I}$*-common* $\vdash_C$ *w-usr* $\surd$
and *pfinite* [*pfinite-intro*]: *pfinite-converter* $\mathcal{I}$*-common*$'$ $\mathcal{I}$*-common w-usr*
and *WT-adv-real* [*WT-intro*]: $\mathcal{I}$*-real*$'$, $\mathcal{I}$*-real* $\vdash_C$ *w-adv-real* $\surd$
and *WT-w-adv-ideal* [*WT-intro*]: $\mathcal{I}$*-ideal*$'$, $\mathcal{I}$*-ideal* $\vdash_C$ *w-adv-ideal* $\surd$
and *WT-adv-ideal-inv* [*WT-intro*]: $\mathcal{I}$*-ideal*, $\mathcal{I}$*-ideal*$'$ $\vdash_C$ *w-adv-ideal-inv* $\surd$
and *ideal-inverse*: $\mathcal{I}$*-ideal*, $\mathcal{I}$*-ideal* $\vdash_C$ *w-adv-ideal-inv* $\odot$ *w-adv-ideal* $\sim$ 1_C
and *pfinite-real* [*pfinite-intro*]: *pfinite-converter* $\mathcal{I}$*-real*$'$ $\mathcal{I}$*-real w-adv-real*
and *pfinite-ideal* [*pfinite-intro*]: *pfinite-converter* $\mathcal{I}$*-ideal* $\mathcal{I}$*-ideal*$'$ *w-adv-ideal-inv*
shows *constructive-security-obsf* (*w-adv-real* $|_=$ *w-usr* $\triangleright$ *real-resource*) (*w-adv-ideal* $|_=$ *w-usr* $\triangleright$ *ideal-resource*) (*w-adv-real* $\odot$ *sim* $\odot$ *w-adv-ideal-inv*) $\mathcal{I}$*-real*$'$ $\mathcal{I}$*-ideal*$'$ $\mathcal{I}$*-common*$'$ $\mathcal{A}$ *adv*
⟨*proof*⟩

theorem *constructive-security-obsf-lifting-usr*:
assumes *sec*: *constructive-security-aux-obsf real-resource ideal-resource sim* $\mathcal{I}$*-real* $\mathcal{I}$*-ideal* $\mathcal{I}$*-common adv*
and *sec2*: *exception-*$\mathcal{I}$ ($\mathcal{I}$*-real* $\oplus_{\mathcal{I}}$ $\mathcal{I}$*-common*$'$) $\vdash g$ $\mathcal{A}$ $\surd$

$\Longrightarrow$ *constructive-security-sim-obsf real-resource ideal-resource sim* $\mathcal{I}$*-real* $\mathcal{I}$*-common* (*absorb* $\mathcal{A}$ (*obsf-converter* (1_C $|_=$ *conv*))) *adv*
and *WT-conv* [*WT-intro*]: $\mathcal{I}$*-common*$'$, $\mathcal{I}$*-common* $\vdash_C$ *conv* $\surd$
and *pfinite* [*pfinite-intro*]: *pfinite-converter* $\mathcal{I}$*-common*$'$ $\mathcal{I}$*-common conv*
shows *constructive-security-obsf* (1_C $|_=$ *conv* $\triangleright$ *real-resource*) (1_C $|_=$ *conv* $\triangleright$ *ideal-resource*) *sim* $\mathcal{I}$*-real* $\mathcal{I}$*-ideal* $\mathcal{I}$*-common*$'$ $\mathcal{A}$ *adv*
⟨*proof*⟩

theorem *constructive-security-obsf-lifting2*:
assumes *sec*: *constructive-security-aux-obsf real-resource ideal-resource sim* ($\mathcal{I}$*-real1* $\oplus_{\mathcal{I}}$ $\mathcal{I}$*-real2*) ($\mathcal{I}$*-ideal1* $\oplus_{\mathcal{I}}$ $\mathcal{I}$*-ideal2*) $\mathcal{I}$*-common adv*
and *sec2*: *exception-*$\mathcal{I}$ (($\mathcal{I}$*-real1* $\oplus_{\mathcal{I}}$ $\mathcal{I}$*-real2*) $\oplus_{\mathcal{I}}$ $\mathcal{I}$*-common*$'$) $\vdash g$ $\mathcal{A}$ $\surd$
$\Longrightarrow$ *constructive-security-sim-obsf real-resource ideal-resource sim* ($\mathcal{I}$*-real1* $\oplus_{\mathcal{I}}$ $\mathcal{I}$*-real2*) $\mathcal{I}$*-common* (*absorb* $\mathcal{A}$ (*obsf-converter* ((1_C $|_=$ 1_C) $|_=$ *conv*))) *adv*
assumes *WT-conv* [*WT-intro*]: $\mathcal{I}$*-common*$'$, $\mathcal{I}$*-common* $\vdash_C$ *conv* $\surd$
and *pfinite* [*pfinite-intro*]: *pfinite-converter* $\mathcal{I}$*-common*$'$ $\mathcal{I}$*-common conv*
shows *constructive-security-obsf* ((1_C $|_=$ 1_C) $|_=$ *conv* $\triangleright$ *real-resource*) ((1_C $|_=$ 1_C) $|_=$ *conv* $\triangleright$ *ideal-resource*) *sim* ($\mathcal{I}$*-real1* $\oplus_{\mathcal{I}}$ $\mathcal{I}$*-real2*) ($\mathcal{I}$*-ideal1* $\oplus_{\mathcal{I}}$ $\mathcal{I}$*-ideal2*) $\mathcal{I}$*-common*$'$ $\mathcal{A}$ *adv*
(**is** *constructive-security-obsf ?real ?ideal - ?*$\mathcal{I}$*-real ?*$\mathcal{I}$*-ideal - - -*)
⟨*proof*⟩

theorem *constructive-security-obsf-trivial*:
fixes *res*
assumes [*WT-intro*]: $\mathcal{I}$ $\oplus_{\mathcal{I}}$ $\mathcal{I}$*-common* $\vdash$*res res* $\surd$
shows *constructive-security-obsf res res* 1_C $\mathcal{I}$ $\mathcal{I}$ $\mathcal{I}$*-common* $\mathcal{A}$ *0*
⟨*proof*⟩

lemma *parallel-constructive-security-aux-obsf* [*locale-witness*]:
assumes *constructive-security-aux-obsf real1 ideal1 sim1* $\mathcal{I}$*-real1* $\mathcal{I}$*-inner1* $\mathcal{I}$*-common1 adv1*
assumes *constructive-security-aux-obsf real2 ideal2 sim2* $\mathcal{I}$*-real2* $\mathcal{I}$*-inner2* $\mathcal{I}$*-common2 adv2*
shows *constructive-security-aux-obsf* (*parallel-wiring* $\triangleright$ *real1* $\parallel$ *real2*) (*parallel-wiring* $\triangleright$ *ideal1* $\parallel$ *ideal2*) (*sim1* $|_=$ *sim2*)
($\mathcal{I}$*-real1* $\oplus_{\mathcal{I}}$ $\mathcal{I}$*-real2*) ($\mathcal{I}$*-inner1* $\oplus_{\mathcal{I}}$ $\mathcal{I}$*-inner2*) ($\mathcal{I}$*-common1* $\oplus_{\mathcal{I}}$ $\mathcal{I}$*-common2*)
(*adv1* + *adv2*)
⟨*proof*⟩

theorem *parallel-constructive-security-obsf*:
assumes *constructive-security-obsf real1 ideal1 sim1* $\mathcal{I}$*-real1* $\mathcal{I}$*-inner1* $\mathcal{I}$*-common1* (*absorb* $\mathcal{A}$ (*obsf-converter* (*parallel-wiring* $\odot$ *parallel-converter* 1_C (*converter-of-resource* (*sim2* $|_=$ 1_C $\triangleright$ *ideal2*))))) *adv1*
(**is** *constructive-security-obsf - - - - - - ?*$\mathcal{A}$*1 -*)
assumes *constructive-security-obsf real2 ideal2 sim2* $\mathcal{I}$*-real2* $\mathcal{I}$*-inner2* $\mathcal{I}$*-common2* (*absorb* $\mathcal{A}$ (*obsf-converter* (*parallel-wiring* $\odot$ *parallel-converter* (*converter-of-resource real1*) 1_C))) *adv2*
(**is** *constructive-security-obsf - - - - - - ?*$\mathcal{A}$*2 -*)
shows *constructive-security-obsf* (*parallel-wiring* $\triangleright$ *real1* $\parallel$ *real2*) (*parallel-wiring*

$\triangleright$ *ideal1* $\parallel$ *ideal2*) (*sim1* $|_=$ *sim2*)
(*$\mathcal{I}$-real1* $\oplus_{\mathcal{I}}$ *$\mathcal{I}$-real2*) (*$\mathcal{I}$-inner1* $\oplus_{\mathcal{I}}$ *$\mathcal{I}$-inner2*) (*$\mathcal{I}$-common1* $\oplus_{\mathcal{I}}$ *$\mathcal{I}$-common2*)
$\mathcal{A}$ (*adv1* + *adv2*)
⟨*proof*⟩

theorem *parallel-constructive-security-obsf-fuse*:
assumes *1*: *constructive-security-obsf real1 ideal1 sim1* (*$\mathcal{I}$-real1-core* $\oplus_{\mathcal{I}}$ *$\mathcal{I}$-real1-rest*) (*$\mathcal{I}$-ideal1-core* $\oplus_{\mathcal{I}}$ *$\mathcal{I}$-ideal1-rest*) (*$\mathcal{I}$-common1-core* $\oplus_{\mathcal{I}}$ *$\mathcal{I}$-common1-rest*) (*absorb* $\mathcal{A}$ (*obsf-converter* (*fused-wiring* $\odot$ *parallel-converter* 1_C (*converter-of-resource* (*sim2* $|_=$ 1_C $\triangleright$ *ideal2*))))) *adv1*
(**is** *constructive-security-obsf - - - ?$\mathcal{I}$-real1 ?$\mathcal{I}$-ideal1 ?$\mathcal{I}$-common1 ?$\mathcal{A}$1 -*)
assumes *2*: *constructive-security-obsf real2 ideal2 sim2* (*$\mathcal{I}$-real2-core* $\oplus_{\mathcal{I}}$ *$\mathcal{I}$-real2-rest*) (*$\mathcal{I}$-ideal2-core* $\oplus_{\mathcal{I}}$ *$\mathcal{I}$-ideal2-rest*) (*$\mathcal{I}$-common2-core* $\oplus_{\mathcal{I}}$ *$\mathcal{I}$-common2-rest*) (*absorb* $\mathcal{A}$ (*obsf-converter* (*fused-wiring* $\odot$ *parallel-converter* (*converter-of-resource real1*) 1_C))) *adv2*
(**is** *constructive-security-obsf - - - ?$\mathcal{I}$-real2 ?$\mathcal{I}$-ideal2 ?$\mathcal{I}$-common2 ?$\mathcal{A}$2 -*)
shows *constructive-security-obsf* (*fused-wiring* $\triangleright$ *real1* $\parallel$ *real2*) (*fused-wiring* $\triangleright$ *ideal1* $\parallel$ *ideal2*)
(*parallel-wiring* $\odot$ (*sim1* $|_=$ *sim2*) $\odot$ *parallel-wiring*)
((*$\mathcal{I}$-real1-core* $\oplus_{\mathcal{I}}$ *$\mathcal{I}$-real2-core*) $\oplus_{\mathcal{I}}$ (*$\mathcal{I}$-real1-rest* $\oplus_{\mathcal{I}}$ *$\mathcal{I}$-real2-rest*))
((*$\mathcal{I}$-ideal1-core* $\oplus_{\mathcal{I}}$ *$\mathcal{I}$-ideal2-core*) $\oplus_{\mathcal{I}}$ (*$\mathcal{I}$-ideal1-rest* $\oplus_{\mathcal{I}}$ *$\mathcal{I}$-ideal2-rest*))
((*$\mathcal{I}$-common1-core* $\oplus_{\mathcal{I}}$ *$\mathcal{I}$-common2-core*) $\oplus_{\mathcal{I}}$ (*$\mathcal{I}$-common1-rest* $\oplus_{\mathcal{I}}$ *$\mathcal{I}$-common2-rest*))
$\mathcal{A}$ (*adv1* + *adv2*)
⟨*proof*⟩

end
theory *Asymptotic-Security* **imports** *Concrete-Security* **begin**

8 Asymptotic security definition

locale *constructive-security-obsf′* =
fixes *real-resource* :: *security* ⇒ (*'a* + *'e*, *'b* + *'f*) *resource*
and *ideal-resource* :: *security* ⇒ (*'c* + *'e*, *'d* + *'f*) *resource*
and *sim* :: *security* ⇒ (*'a*, *'b*, *'c*, *'d*) *converter*
and *$\mathcal{I}$-real* :: *security* ⇒ (*'a*, *'b*) *$\mathcal{I}$*
and *$\mathcal{I}$-ideal* :: *security* ⇒ (*'c*, *'d*) *$\mathcal{I}$*
and *$\mathcal{I}$-common* :: *security* ⇒ (*'e*, *'f*) *$\mathcal{I}$*
and *$\mathcal{A}$* :: *security* ⇒ (*'a* + *'e*, *'b* + *'f*) *distinguisher-obsf*
assumes *constructive-security-aux-obsf*: $\bigwedge\eta$.
constructive-security-aux-obsf (*real-resource* η) (*ideal-resource* η) (*sim* η) (*$\mathcal{I}$-real* η) (*$\mathcal{I}$-ideal* η) (*$\mathcal{I}$-common* η) *0*
and *adv*: ⟦ $\bigwedge\eta$. *exception-$\mathcal{I}$* (*$\mathcal{I}$-real* η $\oplus_{\mathcal{I}}$ *$\mathcal{I}$-common* η) $\vdash g$ $\mathcal{A}$ η $\surd$ ⟧
⟹ *negligible* (λη. *advantage* ($\mathcal{A}$ η) (*obsf-resource* (*sim* η $|_=$ 1_C $\triangleright$ *ideal-resource* η)) (*obsf-resource* (*real-resource* η))))
begin

sublocale *constructive-security-aux-obsf*
real-resource η
ideal-resource η

sim η
$\mathcal{I}$-*real* η
$\mathcal{I}$-*ideal* η
$\mathcal{I}$-*common* η
0
for η ⟨*proof*⟩

lemma *constructive-security-obsf'D*:
constructive-security-obsf (*real-resource* η) (*ideal-resource* η) (*sim* η) ($\mathcal{I}$-*real* η) ($\mathcal{I}$-*ideal* η) ($\mathcal{I}$-*common* η) ($\mathcal{A}$ η)
(*advantage* ($\mathcal{A}$ η) (*obsf-resource* (*sim* η $|_=$ 1_C ▷ *ideal-resource* η)) (*obsf-resource* (*real-resource* η)))
⟨*proof*⟩

end

lemma *constructive-security-obsf'I*:
assumes $\bigwedge\eta$. *constructive-security-obsf* (*real-resource* η) (*ideal-resource* η) (*sim* η) ($\mathcal{I}$-*real* η) ($\mathcal{I}$-*ideal* η) ($\mathcal{I}$-*common* η) ($\mathcal{A}$ η) (*adv* η)
and ($\bigwedge\eta$. *exception-*$\mathcal{I}$ ($\mathcal{I}$-*real* η $\oplus_{\mathcal{I}}$ $\mathcal{I}$-*common* η) $\vdash g$ $\mathcal{A}$ η $\surd$) $\Longrightarrow$ *negligible adv*
shows *constructive-security-obsf'* *real-resource ideal-resource sim* $\mathcal{I}$-*real* $\mathcal{I}$-*ideal* $\mathcal{I}$-*common* $\mathcal{A}$
⟨*proof*⟩

lemma *constructive-security-obsf'-into-constructive-security*:
assumes $\bigwedge\mathcal{A}$ :: *security* ⇒ ('*a* + '*b*, '*c* + '*d*) *distinguisher-obsf*.
⟦ $\bigwedge\eta$. *interaction-bounded-by* (λ-. *True*) ($\mathcal{A}$ η) (*bound* η);
$\bigwedge\eta$. *lossless* $\Longrightarrow$ *plossless-gpv* (*exception-*$\mathcal{I}$ ($\mathcal{I}$-*real* η $\oplus_{\mathcal{I}}$ $\mathcal{I}$-*common* η)) ($\mathcal{A}$ η) ⟧
$\Longrightarrow$ *constructive-security-obsf'* *real-resource ideal-resource sim* $\mathcal{I}$-*real* $\mathcal{I}$-*ideal* $\mathcal{I}$-*common* $\mathcal{A}$
and *correct*: $\exists$ *cnv*. $\forall\mathcal{D}$. ($\forall\eta$. $\mathcal{I}$-*ideal* η $\oplus_{\mathcal{I}}$ $\mathcal{I}$-*common* η $\vdash g$ $\mathcal{D}$ η $\surd$) ⟶
($\forall\eta$. *interaction-any-bounded-by* ($\mathcal{D}$ η) (*bound* η)) ⟶
($\forall\eta$. *lossless* ⟶ *plossless-gpv* ($\mathcal{I}$-*ideal* η $\oplus_{\mathcal{I}}$ $\mathcal{I}$-*common* η) ($\mathcal{D}$ η)) ⟶
($\forall\eta$. *wiring* ($\mathcal{I}$-*ideal* η) ($\mathcal{I}$-*real* η) (*cnv* η) (*w* η)) ∧
Negligible.negligible (λη. *advantage* ($\mathcal{D}$ η) (*ideal-resource* η) (*cnv* η $|_=$ 1_C ▷ *real-resource* η))
shows *constructive-security real-resource ideal-resource sim* $\mathcal{I}$-*real* $\mathcal{I}$-*ideal* $\mathcal{I}$-*common bound lossless w*
⟨*proof*⟩

8.1 Composition theorems

theorem *constructive-security-obsf'-composability*:
fixes *real*
assumes *constructive-security-obsf'* *middle ideal sim-inner* $\mathcal{I}$-*middle* $\mathcal{I}$-*inner* $\mathcal{I}$-*common* (λη. *absorb* ($\mathcal{A}$ η) (*obsf-converter* (*sim-outer* η $|_=$ 1_C)))
assumes *constructive-security-obsf' real middle sim-outer* $\mathcal{I}$-*real* $\mathcal{I}$-*middle* $\mathcal{I}$-*common* $\mathcal{A}$

shows *constructive-security-obsf' real ideal* ($\lambda\eta$. *sim-outer* $\eta \odot$ *sim-inner* η) $\mathcal{I}$*-real* $\mathcal{I}$*-inner* $\mathcal{I}$*-common* $\mathcal{A}$
⟨*proof*⟩

theorem *constructive-security-obsf'-lifting*:
assumes *sec*: *constructive-security-obsf' real-resource ideal-resource sim* $\mathcal{I}$*-real* $\mathcal{I}$*-ideal* $\mathcal{I}$*-common* ($\lambda\eta$. *absorb* ($\mathcal{A}$ η) (*obsf-converter* (1_C $|_=$ *conv* η)))
assumes *WT-conv* [*WT-intro*]: $\bigwedge\eta$. $\mathcal{I}$*-common'* η, $\mathcal{I}$*-common* $\eta \vdash_C$ *conv* η $\surd$
and *pfinite* [*pfinite-intro*]: $\bigwedge\eta$. *pfinite-converter* ($\mathcal{I}$*-common'* η) ($\mathcal{I}$*-common* η) (*conv* η)
shows *constructive-security-obsf'*
($\lambda\eta$. 1_C $|_=$ *conv* $\eta \rhd$ *real-resource* η) ($\lambda\eta$. 1_C $|_=$ *conv* $\eta \rhd$ *ideal-resource* η) *sim* $\mathcal{I}$*-real* $\mathcal{I}$*-ideal* $\mathcal{I}$*-common'* $\mathcal{A}$
⟨*proof*⟩

theorem *constructive-security-obsf'-trivial*:
fixes *res*
assumes [*WT-intro*]: $\bigwedge\eta$. $\mathcal{I}$ $\eta \oplus_{\mathcal{I}}$ $\mathcal{I}$*-common* $\eta \vdash$*res res* η $\surd$
shows *constructive-security-obsf' res res* (λ-. 1_C) $\mathcal{I}$ $\mathcal{I}$ $\mathcal{I}$*-common* $\mathcal{A}$
⟨*proof*⟩

theorem *parallel-constructive-security-obsf'*:
assumes *constructive-security-obsf' real1 ideal1 sim1* $\mathcal{I}$*-real1* $\mathcal{I}$*-inner1* $\mathcal{I}$*-common1* ($\lambda\eta$. *absorb* ($\mathcal{A}$ η) (*obsf-converter* (*parallel-wiring* $\odot$ *parallel-converter* 1_C (*converter-of-resource* (*sim2* η $|_=$ $1_C \rhd$ *ideal2* η)))))
(**is** *constructive-security-obsf' - - - - - - ?*$\mathcal{A}$*1*)
assumes *constructive-security-obsf' real2 ideal2 sim2* $\mathcal{I}$*-real2* $\mathcal{I}$*-inner2* $\mathcal{I}$*-common2* ($\lambda\eta$. *absorb* ($\mathcal{A}$ η) (*obsf-converter* (*parallel-wiring* $\odot$ *parallel-converter* (*converter-of-resource* (*real1* η)) 1_C)))
(**is** *constructive-security-obsf' - - - - - - ?*$\mathcal{A}$*2*)
shows *constructive-security-obsf'* ($\lambda\eta$. *parallel-wiring* $\rhd$ *real1* $\eta \parallel$ *real2* η) ($\lambda\eta$. *parallel-wiring* $\rhd$ *ideal1* $\eta \parallel$ *ideal2* η) ($\lambda\eta$. *sim1* η $|_=$ *sim2* η)
($\lambda\eta$. $\mathcal{I}$*-real1* $\eta \oplus_{\mathcal{I}}$ $\mathcal{I}$*-real2* η) ($\lambda\eta$. $\mathcal{I}$*-inner1* $\eta \oplus_{\mathcal{I}}$ $\mathcal{I}$*-inner2* η) ($\lambda\eta$. $\mathcal{I}$*-common1* $\eta \oplus_{\mathcal{I}}$ $\mathcal{I}$*-common2* η) $\mathcal{A}$
⟨*proof*⟩

end
theory *Key*
imports
../Fused-Resource
begin

9 Key specification

locale *ideal-key* =
fixes *valid-keys* :: *'key set*
begin

9.1 Data-types for Parties, State, Events, Input, and Output

datatype *party* = *Alice* | *Bob*

type-synonym *s-shell* = *party set*
datatype *'key' s-kernel* = *PState-Store* | *State-Store 'key'*
type-synonym *'key' state* = *'key' s-kernel* × *s-shell*

datatype *event* = *Event-Shell party* | *Event-Kernel*

datatype *iadv* = *Inp-Adversary*

datatype *iusr-alice* = *Inp-Alice*
datatype *iusr-bob* = *Inp-Bob*
type-synonym *iusr* = *iusr-alice* + *iusr-bob*

datatype *oadv* = *Out-Adversary*

datatype *'key' ousr-alice* = *Out-Alice 'key'*
datatype *'key' ousr-bob* = *Out-Bob 'key'*
type-synonym *'key' ousr* = *'key' ousr-alice* + *'key' ousr-bob*

9.1.1 Basic lemmas for automated handling of party sets (i.e. *s-shell*)

lemma *Alice-neq-iff* [*simp*]: *Alice* ≠ *x* ⟷ *x* = *Bob*
⟨*proof*⟩

lemma *neq-Alice-iff* [*simp*]: *x* ≠ *Alice* ⟷ *x* = *Bob*
⟨*proof*⟩

lemma *Bob-neq-iff* [*simp*]: *Bob* ≠ *x* ⟷ *x* = *Alice*
⟨*proof*⟩

lemma *neq-Bob-iff* [*simp*]: *x* ≠ *Bob* ⟷ *x* = *Alice*
⟨*proof*⟩

lemma *Alice-in-iff-nonempty*: *Alice* ∈ *A* ⟷ *A* ≠ {} **if** *Bob* ∉ *A*
⟨*proof*⟩

lemma *Bob-in-iff-nonempty*: *Bob* ∈ *A* ⟷ *A* ≠ {} **if** *Alice* ∉ *A*
⟨*proof*⟩

9.2 Defining the event handler

fun *poke* :: (*'key state*, *event*) *handler*
where
poke (*s-kernel*, *parties*) (*Event-Shell party*) =
(*if party* ∈ *parties then*
return-pmf None

```
    else
      return-spmf (s-kernel, insert party parties))
| poke (PState-Store, s-shell) (Event-Kernel) = do {
    key ← spmf-of-set valid-keys;
    return-spmf (State-Store key, s-shell)  }
| poke - - = return-pmf None
```

lemma *in-set-spmf-poke*:
$s' \in$ *set-spmf* (*poke s x*) $\longleftrightarrow$
($\exists$ *s-kernel parties party*. $s =$ (*s-kernel*, *parties*) $\wedge$ $x =$ *Event-Shell party* $\wedge$ *party* $\notin$ *parties* $\wedge$ $s' =$ (*s-kernel*, *insert party parties*)) $\vee$
($\exists$ *s-shell key*. $s =$ (*PState-Store*, *s-shell*) $\wedge$ $x =$ *Event-Kernel* $\wedge$ *key* $\in$ *valid-keys* $\wedge$ *finite valid-keys* $\wedge$ $s' =$ (*State-Store key*, *s-shell*))
⟨*proof*⟩

lemma *foldl-poke-invar*:
⟦ (*s-kernel'*, *parties'*) $\in$ *set-spmf* (*foldl-spmf poke p events*); $\forall$(*s-kernel*, *parties*)$\in$*set-spmf p*. *set-s-kernel s-kernel* $\subseteq$ *valid-keys* ⟧
$\Longrightarrow$ *set-s-kernel s-kernel'* $\subseteq$ *valid-keys*
⟨*proof*⟩

9.3 Defining the adversary interface

fun *iface-adv* :: (*'key state*, *iadv*, *oadv*) *oracle'*
where
iface-adv state - = return-spmf (*Out-Adversary*, *state*)

9.4 Defining the user interfaces

context
begin

private fun *iface-usr-func* :: *party* ⇒ - ⇒ - ⇒ *'inp* ⇒ (*'wrap-key* × *'key state*) *spmf*
where
```
  iface-usr-func party wrap (State-Store key, parties) inp =
    (if party ∈ parties then
      return-spmf (wrap key, State-Store key, parties)
    else
      return-pmf None)
| iface-usr-func - - - - = return-pmf None
```

abbreviation *iface-alice* :: (*'key state*, *iusr-alice*, *'key ousr-alice*) *oracle'*
where
iface-alice ≡ *iface-usr-func Alice Out-Alice*

abbreviation *iface-bob* :: (*'key state*, *iusr-bob*, *'key ousr-bob*) *oracle'*
where
iface-bob ≡ *iface-usr-func Bob Out-Bob*

abbreviation *iface-usr* :: (*'key state*, *iusr*, *'key ousr*) *oracle'*
where
iface-usr ≡ *plus-oracle iface-alice iface-bob*

lemma *in-set-iface-usr-func* [*simp*]:
x ∈ *set-spmf* (*iface-usr-func party wrap state inp*) ⟷
(∃ *key parties*. *state* = (*State-Store key*, *parties*) ∧ *party* ∈ *parties* ∧ *x* = (*wrap key*, *State-Store key*, *parties*))
⟨*proof*⟩

end

9.5 Defining the Fuse Resource

primcorec *core* :: (*'key state*, *event*, *iadv*, *iusr*, *oadv*,*'key ousr*) *core*
where
cpoke core = *poke*
| *cfunc-adv core* = *iface-adv*
| *cfunc-usr core* = *iface-usr*

sublocale *fused-resource core* (*PState-Store*, {}) ⟨*proof*⟩

9.5.1 Lemma showing that the resulting resource is well-typed

lemma *WT-core* [*WT-intro*]:
WT-core $\mathcal{I}$*-full* ($\mathcal{I}$*-uniform UNIV* (*Out-Alice* ‘ *valid-keys*) $\oplus_{\mathcal{I}}$ $\mathcal{I}$*-uniform UNIV* (*Out-Bob* ‘ *valid-keys*))
(*pred-prod* (*pred-s-kernel* (λ*key*. *key* ∈ *valid-keys*)) (λ-. *True*)) *core*
⟨*proof*⟩

lemma *WT-fuse* [*WT-intro*]:
assumes [*WT-intro*]: *WT-rest* $\mathcal{I}$*-adv-rest* $\mathcal{I}$*-usr-rest I-rest rest*
shows ($\mathcal{I}$*-full* $\oplus_{\mathcal{I}}$ $\mathcal{I}$*-adv-rest*) $\oplus_{\mathcal{I}}$ (($\mathcal{I}$*-uniform UNIV* (*Out-Alice* ‘ *valid-keys*) $\oplus_{\mathcal{I}}$ $\mathcal{I}$*-uniform UNIV* (*Out-Bob* ‘ *valid-keys*)) $\oplus_{\mathcal{I}}$ $\mathcal{I}$*-usr-rest*) ⊢*res resource rest* √
⟨*proof*⟩

end

end
theory *Channel*
imports
../Fused-Resource
begin

10 Channel specification

locale *ideal-channel* =
fixes

leak :: *'msg* ⇒ *'leak* **and**
editable :: *bool*
begin

10.1 Data-types for Parties, State, Events, Input, and Output

datatype *party* = *Alice* | *Bob*

type-synonym *s-shell* = *party set*
datatype *'msg' s-kernel* = *State-Void* | *State-Store 'msg'* | *State-Collect 'msg'* | *State-Collected*
type-synonym *'msg' state* = *'msg' s-kernel* × *s-shell*

datatype *event* = *Event-Shell party*

datatype *iadv-drop* = *Inp-Drop*
datatype *iadv-look* = *Inp-Look*
datatype *'msg' iadv-fedit* = *Inp-Fedit 'msg'*
type-synonym *'msg' iadv* = *iadv-drop* + *iadv-look* + *'msg' iadv-fedit*

datatype *'msg' iusr-alice* = *Inp-Send 'msg'*
datatype *iusr-bob* = *Inp-Recv*
type-synonym *'msg' iusr* = *'msg' iusr-alice* + *iusr-bob*

datatype *oadv-drop* = *Out-Drop*
datatype *'leak' oadv-look* = *Out-Look 'leak'*
datatype *oadv-fedit* = *Out-Fedit*
type-synonym *'leak' oadv* = *oadv-drop* + *'leak' oadv-look* + *oadv-fedit*

datatype *ousr-alice* = *Out-Send*
datatype *'msg' ousr-bob* = *Out-Recv 'msg'*
type-synonym *'msg' ousr* = *ousr-alice* + *'msg' ousr-bob*

10.1.1 Basic lemmas for automated handling of party sets (i.e. *s-shell*)

lemma *Alice-neq-iff* [*simp*]: *Alice* ≠ *x* ⟷ *x* = *Bob*
⟨*proof*⟩

lemma *neq-Alice-iff* [*simp*]: *x* ≠ *Alice* ⟷ *x* = *Bob*
⟨*proof*⟩

lemma *Bob-neq-iff* [*simp*]: *Bob* ≠ *x* ⟷ *x* = *Alice*
⟨*proof*⟩

lemma *neq-Bob-iff* [*simp*]: *x* ≠ *Bob* ⟷ *x* = *Alice*
⟨*proof*⟩

lemma *Alice-in-iff-nonempty*: $Alice \in A \longleftrightarrow A \neq \{\}$ **if** $Bob \notin A$
⟨*proof*⟩

lemma *Bob-in-iff-nonempty*: $Bob \in A \longleftrightarrow A \neq \{\}$ **if** $Alice \notin A$
⟨*proof*⟩

10.2 Defining the event handler

fun *poke* :: (*'msg state*, *event*) *handler*
where
poke (*s-kernel*, *parties*) (*Event-Shell party*) =
(*if party* ∈ *parties then*
return-pmf None
else
return-spmf (*s-kernel*, *insert party parties*))

lemma *poke-alt-def*:
poke = (λ(*s*, *ps*) *e*. *map-spmf* (*Pair s*) (*case e of Event-Shell party* ⇒ *if party* ∈ *ps then return-pmf None else return-spmf* (*insert party ps*)))
⟨*proof*⟩

10.3 Defining the adversary interfaces

fun *iface-drop* :: (*'msg state*, *iadv-drop*, *oadv-drop*) *oracle'*
where
iface-drop - - = *return-pmf None*

fun *iface-look* :: (*'msg state*, *iadv-look*, *'leak oadv-look*) *oracle'*
where
iface-look (*State-Store msg*, *parties*) - =
return-spmf (*Out-Look* (*leak msg*), *State-Store msg*, *parties*)
| *iface-look* - - = *return-pmf None*

fun *iface-fedit* :: (*'msg state*, *'msg iadv-fedit*, *oadv-fedit*) *oracle'*
where
iface-fedit (*State-Store msg*, *parties*) (*Inp-Fedit msg'*) =
(*if editable then*
return-spmf (*Out-Fedit*, *State-Collect msg'*, *parties*)
else
return-spmf (*Out-Fedit*, *State-Collect msg*, *parties*))
| *iface-fedit* - - = *return-pmf None*

abbreviation *iface-adv* :: (*'msg state*, *'msg iadv*, *'leak oadv*) *oracle'*
where
iface-adv ≡ *plus-oracle iface-drop* (*plus-oracle iface-look iface-fedit*)

lemma *in-set-spmf-iface-drop*: $ys' \in$ *set-spmf* (*iface-drop s x*) $\longleftrightarrow$ *False*
⟨*proof*⟩

lemma *in-set-spmf-iface-look*: *ys′* ∈ *set-spmf* (*iface-look s x*) ⟷
(∃ *msg parties*. *s* = (*State-Store msg*, *parties*) ∧ *ys′* = (*Out-Look* (*leak msg*), *State-Store msg*, *parties*))
⟨*proof*⟩

lemma *in-set-spmf-iface-fedit*: *ys′* ∈ *set-spmf* (*iface-fedit s x*) ⟷
(∃ *msg parties msg′*. *s* = (*State-Store msg*, *parties*) ∧ *x* = (*Inp-Fedit msg′*) ∧
ys′ = (*if editable then* (*Out-Fedit*, *State-Collect msg′*, *parties*) *else* (*Out-Fedit*, *State-Collect msg*, *parties*)))
⟨*proof*⟩

10.4 Defining the user interfaces

fun *iface-alice* :: (*′msg state*, *′msg iusr-alice*, *ousr-alice*) *oracle′*
where
iface-alice (*State-Void*, *parties*) (*Inp-Send msg*) =
(*if Alice* ∈ *parties then*
return-spmf (*Out-Send*, *State-Store msg*, *parties*)
else
return-pmf None)
| *iface-alice* - - = *return-pmf None*

fun *iface-bob* :: (*′msg state*, *iusr-bob*, *′msg ousr-bob*) *oracle′*
where
iface-bob (*State-Collect msg*, *parties*) - =
(*if Bob* ∈ *parties then*
return-spmf (*Out-Recv msg*, *State-Collected*, *parties*)
else
return-pmf None)
| *iface-bob* - - = *return-pmf None*

abbreviation *iface-usr* :: (*′msg state*, *′msg iusr*, *′msg ousr*) *oracle′*
where
iface-usr ≡ *plus-oracle iface-alice iface-bob*

lemma *in-set-spmf-iface-alice*: *ys′* ∈ *set-spmf* (*iface-alice s x*) ⟷
(∃ *parties msg*. *s* = (*State-Void*, *parties*) ∧ *x* = *Inp-Send msg* ∧ *Alice* ∈ *parties* ∧ *ys′* = (*Out-Send*, *State-Store msg*, *parties*))
⟨*proof*⟩

lemma *in-set-spmf-iface-bob*: *ys′* ∈ *set-spmf* (*iface-bob s x*) ⟷
(∃ *msg parties*. *s* = (*State-Collect msg*, *parties*) ∧ *Bob* ∈ *parties* ∧ *ys′* = (*Out-Recv msg*, *State-Collected*, *parties*))
⟨*proof*⟩

10.5 Defining the Fused Resource

primcorec *core* :: (*′msg state*, *event*, *′msg iadv*, *′msg iusr*, *′leak oadv*, *′msg ousr*) *core*
where

cpoke core = poke
| *cfunc-adv core = iface-adv*
| *cfunc-usr core = iface-usr*

sublocale *fused-resource core* (*State-Void*, {}) ⟨*proof*⟩

10.5.1 Lemma showing that the resulting resource is well-typed

lemma *WT-core* [*WT-intro*]:
WT-core (*I-full* $\oplus_{\mathcal{I}}$ (*I-full* $\oplus_{\mathcal{I}}$ *I-uniform* (*Inp-Fedit* ‘ *valid-messages*) *UNIV*)) (*I-uniform* (*Inp-Send* ‘ *valid-messages*) *UNIV* $\oplus_{\mathcal{I}}$ (*I-uniform UNIV* (*Out-Recv* ‘ *valid-messages*)))
(*pred-prod* (*pred-s-kernel* (λ*msg. msg* ∈ *valid-messages*)) (λ-. *True*)) *core*
⟨*proof*⟩

lemma *WT-fuse* [*WT-intro*]:
assumes [*WT-intro*]: *WT-rest I-adv-rest I-usr-rest I-rest rest*
shows ((*I-full* $\oplus_{\mathcal{I}}$ (*I-full* $\oplus_{\mathcal{I}}$ *I-uniform* (*Inp-Fedit* ‘ *valid-messages*) *UNIV*)) $\oplus_{\mathcal{I}}$ *I-adv-rest*) $\oplus_{\mathcal{I}}$
((*I-uniform* (*Inp-Send* ‘ *valid-messages*) *UNIV* $\oplus_{\mathcal{I}}$ *I-uniform UNIV* (*Out-Recv* ‘ *valid-messages*)) $\oplus_{\mathcal{I}}$ *I-usr-rest*) ⊢*res resource rest* √
⟨*proof*⟩

end

end
theory *One-Time-Pad*
imports
Sigma-Commit-Crypto.Xor
../Asymptotic-Security
../Construction-Utility
../Specifications/Key
../Specifications/Channel
begin

11 One-time-pad construction

locale *one-time-pad* =
key: *ideal-key carrier* $\mathcal{L}$ +
auth: *ideal-channel id* :: *'msg* ⇒ *'msg False* +
sec: *ideal-channel* λ- :: *'msg. carrier* $\mathcal{L}$ *False* +
boolean-algebra $\mathcal{L}$
for
$\mathcal{L}$ :: (*'msg*, *'more*) *boolean-algebra-scheme* (**structure**) +
assumes
nempty-carrier: *carrier* $\mathcal{L}$ ≠ {} **and**
finite-carrier: *finite* (*carrier* $\mathcal{L}$)
begin

11.1 Defining user callees

definition *enc-callee* :: *unit* ⇒ *'msg sec.iusr-alice*
⇒ (*sec.ousr-alice* × *unit*, *key.iusr-alice* + *'msg sec.iusr-alice*, *'msg key.ousr-alice* + *auth.ousr-alice*) *gpv*
where
enc-callee ≡ *stateless-callee* (λ*inp*. *case inp of sec.Inp-Send msg* ⇒
if msg ∈ *carrier* 𝓛 *then*
Pause
(*Inl key.Inp-Alice*)
(λ*kout*. *case projl kout of key.Out-Alice key* ⇒
let cipher = *key* ⊕ *msg in*
Pause (*Inr* (*auth.Inp-Send cipher*)) (λ-. *Done sec.Out-Send*))
else
Fail)

definition *dec-callee* :: *unit* ⇒ *sec.iusr-bob*
⇒ (*'msg sec.ousr-bob* × *unit*, *key.iusr-bob* + *auth.iusr-bob*, *'msg key.ousr-bob* + *'msg auth.ousr-bob*) *gpv*
where
dec-callee ≡ *stateless-callee* (λ-.
Pause
(*Inr auth.Inp-Recv*)
(λ*cout*. *case cout of*
Inr (*auth.Out-Recv cipher*) ⇒
Pause
(*Inl key.Inp-Bob*)
(λ*kout*. *case projl kout of key.Out-Bob key* ⇒
Done (*sec.Out-Recv* (*key* ⊕ *cipher*)))
| - ⇒ *Fail*))

11.2 Defining adversary converter

type-synonym *'msg' astate* = *'msg' option*

definition *look-callee* :: *'msg astate* ⇒ *sec.iadv-look*
⇒ (*'msg sec.oadv-look* × *'msg astate*, *sec.iadv-look*, *'msg set sec.oadv-look*) *gpv*
where
look-callee ≡ λ*state inp*.
Pause
sec.Inp-Look
(λ*cout*. *case cout of*
sec.Out-Look msg-set ⇒
(*case state of*
None ⇒ *do* {
msg ← *lift-spmf* (*spmf-of-set* (*msg-set*));
Done (*auth.Out-Look msg*, *Some msg*) }
| *Some msg* ⇒ *Done* (*auth.Out-Look msg*, *Some msg*)))

definition *sim* ::

```
  (key.iadv + auth.iadv-drop + auth.iadv-look + 'msg auth.iadv-fedit,
    key.oadv + auth.oadv-drop + 'msg auth.oadv-look + auth.oadv-fedit,
    sec.iadv-drop + sec.iadv-look + 'msg sec.iadv-fedit,
    sec.oadv-drop + 'msg set sec.oadv-look + sec.oadv-fedit) converter
  where
    sim ≡
      let look-converter = converter-of-callee look-callee None in
      ldummy-converter (λ-. key.Out-Adversary) (1_C |_= look-converter |_= 1_C)
```

11.3 Defining event-translator

```
type-synonym estate = bool × (key.party + auth.party) set

abbreviation einit :: estate
  where
    einit ≡ (False, {})

definition sec-party-of-key-party :: key.party ⇒ sec.party
  where
    sec-party-of-key-party ≡ key.case-party sec.Alice sec.Bob

abbreviation etran-base-helper :: estate ⇒ key.party + auth.party ⇒ sec.event
list
  where
    etran-base-helper ≡ (λ(s-flg, s-kap) item.
      let sp-of = case-sum sec-party-of-key-party id in
      let se-of = (λchk out. if s-flg ∧ chk then [out] else []) in
      let chk-alice = Inl key.Alice ∈ s-kap ∧ Inr auth.Alice ∈ s-kap in
      let chk-bob = Inl key.Bob ∈ s-kap ∧ Inr auth.Bob ∈ s-kap in
      sec.case-party
        (se-of chk-alice (sec.Event-Shell sec.Alice))
        (se-of chk-bob (sec.Event-Shell sec.Bob))
        (sp-of item))

abbreviation etran-base :: (estate, key.party + auth.party, sec.event list) oracle'
  where
    etran-base ≡ (λ(s-flg, s-kap) item.
      let s-kap' = insert item s-kap in
      let event = etran-base-helper (s-flg, s-kap') item in
      if item ∉ s-kap then return-spmf (event, s-flg, s-kap') else return-pmf None)

fun etran :: (estate, key.event + auth.event, sec.event list) oracle'
  where
    etran state (Inl (key.Event-Shell party)) = etran-base state (Inl party)
  | etran (False, s-kap) (Inl key.Event-Kernel) =
      (let check-alice = Inl key.Alice ∈ s-kap ∧ Inr auth.Alice ∈ s-kap in
      let check-bob = Inl key.Bob ∈ s-kap ∧ Inr auth.Bob ∈ s-kap in
      let e-alice = if check-alice then [sec.Event-Shell sec.Alice] else [] in
      let e-bob = if check-bob then [sec.Event-Shell sec.Bob] else [] in
```

return-spmf (*e-alice* @ *e-bob*, *True*, *s-kap*))
| *etran state* (*Inr* (*auth.Event-Shell party*)) = *etran-base state* (*Inr party*)
| *etran* - - = *return-pmf None*

11.3.1 Basic lemmas for automated handling of *sec-party-of-key-party*

lemma *sec-party-of-key-party-simps* [*simp*]:
sec-party-of-key-party key.Alice = *sec.Alice*
sec-party-of-key-party key.Bob = *sec.Bob*
⟨*proof*⟩

lemma *sec-party-of-key-party-eq-simps* [*simp*]:
sec-party-of-key-party p = *sec.Alice* ⟷ *p* = *key.Alice*
sec-party-of-key-party p = *sec.Bob* ⟷ *p* = *key.Bob*
⟨*proof*⟩

lemma *key-case-party-collapse* [*simp*]: *key.case-party x x p* = *x*
⟨*proof*⟩

lemma *sec-case-party-collapse* [*simp*]: *sec.case-party x x p* = *x*
⟨*proof*⟩

lemma *Alice-in-sec-party-of-key-party* [*simp*]:
sec.Alice ∈ *sec-party-of-key-party* ‘ *P* ⟷ *key.Alice* ∈ *P*
⟨*proof*⟩

lemma *Bob-in-sec-party-of-key-party* [*simp*]:
sec.Bob ∈ *sec-party-of-key-party* ‘ *P* ⟷ *key.Bob* ∈ *P*
⟨*proof*⟩

lemma *case-sec-party-of-key-party* [*simp*]: *sec.case-party a b* (*sec-party-of-key-party x*) = *key.case-party a b x*
⟨*proof*⟩

11.4 Defining Ideal and Real constructions

context
fixes
key-rest :: (*'key-s-rest*, *key.event*, *'key-iadv-rest*, *'key-iusr-rest*, *'key-oadv-rest*, *'key-ousr-rest*) *rest-wstate* **and**
auth-rest :: (*'auth-s-rest*, *auth.event*, *'auth-iadv-rest*, *'auth-iusr-rest*, *'auth-oadv-rest*, *'auth-ousr-rest*) *rest-wstate*
begin

definition *ideal-rest*
where
ideal-rest ≡ *translate-rest einit etran* (*parallel-rest key-rest auth-rest*)

definition *ideal-resource*
where

ideal-resource ≡ (*sim* |= 1_C) |= 1_C |= 1_C ▷ (*sec.resource ideal-rest*)

definition *real-resource*
where
real-resource ≡ *attach-c1f22-c1f22* (*CNV enc-callee* ()) (*CNV dec-callee* ())
(*key.resource key-rest*) (*auth.resource auth-rest*)

11.5 Wiring and simplifying the Ideal construction

definition *ideal-s-core′* :: ((- × *'msg astate* × -) × -) × *estate* × *'msg sec.state*
where
ideal-s-core′ ≡ ((((), *None*, ()), ()), (*False*, {}), *sec.State-Void*, {})

definition *ideal-s-rest′* :: - × *'key-s-rest* × *'auth-s-rest*
where
ideal-s-rest′ ≡ (((), ()), *rinit key-rest*, *rinit auth-rest*)

primcorec *ideal-core′* :: (((*unit* × - × *unit*) × *unit*) × -, -, *key.iadv* + -, -, -, -)
core
where
cpoke ideal-core′ = (λ(*s-advusr*, *s-event*, *s-core*) *event*. *do* {
(*events*, *s-event′*) ← (*etran s-event event*);
s-core′ ← *foldl-spmf sec.poke* (*return-spmf s-core*) *events*;
return-spmf (*s-advusr*, *s-event′*, *s-core′*)
})
| *cfunc-adv ideal-core′* = (λ((*s-adv*, *s-usr*), *s-core*) *iadv*.
let handle-l = (λ-. *Done* (*Inl key.Out-Adversary*, *s-adv*)) *in*
let handle-r = (λ*qr*. *map-gpv* (*map-prod Inr id*) *id* ((1_I ‡$_I$ *look-callee* ‡$_I$ 1_I)
s-adv qr)) *in*
map-spmf
(λ((*oadv*, *s-adv′*), *s-core′*). (*oadv*, (*s-adv′*, *s-usr*), *s-core′*))
(*exec-gpv* †*sec.iface-adv* (*case-sum handle-l handle-r iadv*) *s-core*))
| *cfunc-usr ideal-core′* = ††*sec.iface-usr*

primcorec *ideal-rest′* :: ((*unit* × *unit*) × -, -, -, -, -, -, -) *rest-scheme*
where
rinit ideal-rest′ = (((), ()), *rinit key-rest*, *rinit auth-rest*)
| *rfunc-adv ideal-rest′* = †(*parallel-eoracle* (*rfunc-adv key-rest*) (*rfunc-adv auth-rest*))
| *rfunc-usr ideal-rest′* = †(*parallel-eoracle* (*rfunc-usr key-rest*) (*rfunc-usr auth-rest*))

11.5.1 The ideal attachment lemma

lemma *attach-ideal*: *ideal-resource* = *RES* (*fused-resource.fuse ideal-core′ ideal-rest′*)
(*ideal-s-core′*, *ideal-s-rest′*)
⟨*proof*⟩

11.6 Wiring and simplifying the Real construction

definition *real-s-core′* :: - × *'msg key.state* × *'msg auth.state*
where

real-s-core′ ≡ ((((), (), ()), (*key.PState-Store*, {}), (*auth.State-Void*, {}))

definition *real-s-rest′*
where
real-s-rest′ ≡ *ideal-s-rest′*

primcorec *real-core′* :: ((*unit* × -) × -, -, -, -, -, -) *core*
where
cpoke real-core′ = (λ(*s-advusr*, *s-core*) *event*.
map-spmf (*Pair s-advusr*) (*parallel-handler key.poke auth.poke s-core event*))
| *cfunc-adv real-core′* = †(*key.iface-adv* ‡$_O$ *auth.iface-adv*)
| *cfunc-usr real-core′* = (λ((*s-adv*, *s-usr*), *s-core*) *iusr*.
let handle-req = *lsumr* ∘ *map-sum id* (*rsuml* ∘ *map-sum swap-sum id* ∘ *lsumr*) ∘ *rsuml in*
let handle-ret = *lsumr* ∘ (*map-sum id* (*rsuml* ∘ (*map-sum swap-sum id* ∘ *lsumr*)) ∘ *rsuml*) *in*
map-spmf
(λ((*ousr*, *s-usr′*), *s-core′*). (*ousr*, (*s-adv*, *s-usr′*), *s-core′*))
(*exec-gpv*
(*key.iface-usr* ‡$_O$ *auth.iface-usr*)
(*map-gpv′ id handle-req handle-ret* ((*enc-callee* ‡$_I$ *dec-callee*) *s-usr iusr*)) *s-core*))

definition *real-rest′*
where
real-rest′ ≡ *ideal-rest′*

11.6.1 The real attachment lemma

private lemma *WT-callee-real1*: ((*I-full* ⊕$_\mathcal{I}$ *I-full*) ⊕$_\mathcal{I}$ (*I-full* ⊕$_\mathcal{I}$ *I-full*)) ⊕$_\mathcal{I}$ ((*I-full* ⊕$_\mathcal{I}$ *I-full*) ⊕$_\mathcal{I}$ (*I-full* ⊕$_\mathcal{I}$ *I-full*)) ⊢ *c*
(*key.fuse key-rest* ‡$_O$ *auth.fuse auth-rest*) *s* √
⟨*proof*⟩ **lemma** *WT-callee-real2*: (*I-full* ⊕$_\mathcal{I}$ *I-full*) ⊕$_\mathcal{I}$ (((*I-full* ⊕$_\mathcal{I}$ *I-full*) ⊕$_\mathcal{I}$ (*I-full* ⊕$_\mathcal{I}$ *I-full*)) ⊕$_\mathcal{I}$ *I-full*) ⊢ *c*
fused-resource.fuse (*parallel-core key.core auth.core*) (*parallel-rest key-rest auth-rest*) *s* √
⟨*proof*⟩

lemma *attach-real*: *real-resource* = *RES* (*fused-resource.fuse real-core′ real-rest′*) (*real-s-core′*, *real-s-rest′*)
⟨*proof*⟩

11.7 Proving the trace-equivalence of simplified Ideal and Real constructions

context
begin

11.7.1 Proving the trace-equivalence of cores

private abbreviation
a-I ≡ λ(*x*, *y*). ((((), *x*, ()), ()), *y*)

private abbreviation
a-R ≡ λ*x*. (((), (), ()), *x*)

abbreviation
asm-act ≡ (λ*flg pset-sec pset-key pset-auth pset-union*.
pset-union = *pset-key* <+> *pset-auth* ∧
(*flg* ⟶ *pset-sec* = *sec-party-of-key-party* ‘ *pset-key* ∩ *pset-auth*))

private inductive *S* :: (((- × *'msg option* × -) × -) × *estate* × *'msg sec.state*) *spmf*
⇒ (- × *'msg key.state* × *'msg auth.state*) *spmf* ⇒ *bool*
where
— (Auth =a)@(Key =0)
s-0-0: *S* (*return-spmf* (*a-I* (*None*, (*False*, *s-act-ka*), *sec.State-Void*, *s-act-s*)))
(*return-spmf* (*a-R* ((*key.PState-Store*, *s-act-k*), *auth.State-Void*, *s-act-a*)))
if *asm-act False s-act-s s-act-k s-act-a s-act-ka* **and** *s-act-s* = {}
— (Auth =a)@(Key =1)
| *s-0-1*: *S* (*return-spmf* (*a-I* (*None*, (*True*, *s-act-ka*), *sec.State-Void*, *s-act*)))
(*map-spmf* (λ*key*. *a-R* ((*key.State-Store key*, *s-act-k*), *auth.State-Void*, *s-act-a*)) (*spmf-of-set* (*carrier* $\mathcal{L}$)))
if *asm-act True s-act s-act-k s-act-a s-act-ka*
— ../(Auth =a)@(Key =1) # wl
| *s-1-1*: *S* (*return-spmf* (*a-I* (*None*, (*True* ,*s-act-ka*), *sec.State-Store msg*, *s-act-s*)))
(*map-spmf* (λ*key*. *a-R* ((*key.State-Store key*, *s-act-k*), *auth.State-Store* (*key* ⊕ *msg*), *s-act-a*)) (*spmf-of-set* (*carrier* $\mathcal{L}$)))
if *asm-act True s-act-s s-act-k s-act-a s-act-ka* **and** *key.Alice* ∈ *s-act-k* **and** *auth.Alice* ∈ *s-act-a* **and** *msg* ∈ *carrier* $\mathcal{L}$
| *s-2-1*: *S* (*return-spmf* (*a-I* (*None*, (*True* ,*s-act-ka*), *sec.State-Collect msg*, *s-act-s*)))
(*map-spmf* (λ*key*. *a-R* ((*key.State-Store key*, *s-act-k*), *auth.State-Collect* (*key* ⊕ *msg*), *s-act-a*)) (*spmf-of-set* (*carrier* $\mathcal{L}$)))
if *asm-act True s-act-s s-act-k s-act-a s-act-ka* **and** *key.Alice* ∈ *s-act-k* **and** *auth.Alice* ∈ *s-act-a* **and** *msg* ∈ *carrier* $\mathcal{L}$
| *s-3-1*: *S* (*return-spmf* (*a-I* (*None*, (*True* ,*s-act-ka*), *sec.State-Collected*, *s-act-s*)))
(*map-spmf* (λ*key*. *a-R* ((*key.State-Store key*, *s-act-k*), *auth.State-Collected*, *s-act-a*)) (*spmf-of-set* (*carrier* $\mathcal{L}$)))
if *asm-act True s-act-s s-act-k s-act-a s-act-ka* **and** *s-act-k* = {*key.Alice*, *key.Bob*} **and** *s-act-a* = {*auth.Alice*, *auth.Bob*}
— ../(Auth =a)@(Key =1) # look
| *s-1′-1*: *S* (*return-spmf* (*a-I* (*Some* (*key* ⊕ *msg*), (*True* ,*s-act-ka*), *sec.State-Store msg*, *s-act-s*)))
(*return-spmf* (*a-R* ((*key.State-Store key*, *s-act-k*), *auth.State-Store* (*key* ⊕ *msg*), *s-act-a*)))
if *asm-act True s-act-s s-act-k s-act-a s-act-ka* **and** *key.Alice* ∈ *s-act-k* **and** *auth.Alice* ∈ *s-act-a* **and** *msg* ∈ *carrier* $\mathcal{L}$ **and** *key* ∈ *carrier* $\mathcal{L}$

| *s-2′-1*: *S* (*return-spmf* (*a-I* (*Some* (*key* ⊕ *msg*), (*True* ,*s-act-ka*), *sec.State-Collect* *msg*, *s-act-s*)))
(*return-spmf* (*a-R* ((*key.State-Store key*, *s-act-k*), *auth.State-Collect* (*key* ⊕ *msg*), *s-act-a*)))
if *asm-act True s-act-s s-act-k s-act-a s-act-ka* **and** *key.Alice* ∈ *s-act-k* **and** *auth.Alice* ∈ *s-act-a* **and** *msg* ∈ *carrier* $\mathcal{L}$ **and** *key* ∈ *carrier* $\mathcal{L}$
| *s-3′-1*: *S* (*return-spmf* (*a-I* (*Some* (*key* ⊕ *msg*), (*True* ,*s-act-ka*), *sec.State-Collected*, *s-act-s*)))
(*return-spmf* (*a-R* ((*key.State-Store key*, *s-act-k*), *auth.State-Collected*, *s-act-a*)))
if *asm-act True s-act-s s-act-k s-act-a s-act-ka* **and** *s-act-k* = {*key.Alice*, *key.Bob*} **and** *s-act-a* = {*auth.Alice*, *auth.Bob*} **and** *msg* ∈ *carrier* $\mathcal{L}$ **and** *key* ∈ *carrier* $\mathcal{L}$

private lemma *trace-eq-core*: *trace-core-eq ideal-core′ real-core′*
UNIV (*UNIV* <+> *UNIV* <+> *UNIV* <+> (*auth.Inp-Fedit* ‘ *carrier* $\mathcal{L}$)) ((*sec.Inp-Send* ‘ *carrier* $\mathcal{L}$) <+> *UNIV*)
(*return-spmf ideal-s-core′*) (*return-spmf real-s-core′*)
⟨*proof*⟩

11.7.2 Proving the trace equivalence of fused cores and rests

private definition $\mathcal{I}$*-adv-core* :: (*key.iadv* + *′msg auth.iadv*, *key.oadv* + *′msg auth.oadv*) $\mathcal{I}$
where $\mathcal{I}$*-adv-core* ≡ $\mathcal{I}$*-full* $\oplus_{\mathcal{I}}$ ($\mathcal{I}$*-full* $\oplus_{\mathcal{I}}$ $\mathcal{I}$*-uniform* (*sec.Inp-Fedit* ‘ (*carrier* $\mathcal{L}$)) *UNIV*))

private definition $\mathcal{I}$*-usr-core* :: (*′msg sec.iusr*, *′msg sec.ousr*) $\mathcal{I}$
where $\mathcal{I}$*-usr-core* ≡ $\mathcal{I}$*-uniform* (*sec.Inp-Send* ‘ (*carrier* $\mathcal{L}$)) *UNIV* $\oplus_{\mathcal{I}}$ $\mathcal{I}$*-uniform* *UNIV* (*sec.Out-Recv* ‘ *carrier* $\mathcal{L}$)

private definition *invar-ideal′* :: ((- × *′msg astate* × -) × -) × *estate* × *′msg sec.state* ⇒ *bool*
where *invar-ideal′* = *pred-prod* (*pred-prod* (*pred-prod* (λ-. *True*) (*pred-prod* (*pred-option* (λ*x*. *x* ∈ *carrier* $\mathcal{L}$)) (λ-. *True*))) (λ-. *True*)) (*pred-prod* (λ-. *True*) (*pred-prod* (*sec.pred-s-kernel* (λ*x*. *x* ∈ *carrier* $\mathcal{L}$)) (λ-. *True*)))

private definition *invar-real′* :: - × (*′msg key.s-kernel* × -) × *′msg sec.s-kernel* × - ⇒ *bool*
where *invar-real′* = *pred-prod* (λ-. *True*) (*pred-prod* (*pred-prod* (*key.pred-s-kernel* (λ*x*. *x* ∈ *carrier* $\mathcal{L}$)) (λ-. *True*)) (*pred-prod* (*sec.pred-s-kernel* (λ*x*. *x* ∈ *carrier* $\mathcal{L}$)) (λ-. *True*)))

lemma *invar-ideal-s-core′* [*simp*]: *invar-ideal′ ideal-s-core′*
⟨*proof*⟩

lemma *invar-real-s-core′* [*simp*]: *invar-real′ real-s-core′*
⟨*proof*⟩

lemma *WT-ideal-core′* [*WT-intro*]: *WT-core* $\mathcal{I}$*-adv-core* $\mathcal{I}$*-usr-core invar-ideal′ ideal-core′*

⟨*proof*⟩

lemma *WT-ideal-rest′* [*WT-intro*]:
assumes *WT-rest* $\mathcal{I}$*-adv-restk* $\mathcal{I}$*-usr-restk I-key-rest key-rest*
and *WT-rest* $\mathcal{I}$*-adv-resta* $\mathcal{I}$*-usr-resta I-auth-rest auth-rest*
shows *WT-rest* ($\mathcal{I}$*-adv-restk* $\oplus_{\mathcal{I}}$ $\mathcal{I}$*-adv-resta*) ($\mathcal{I}$*-usr-restk* $\oplus_{\mathcal{I}}$ $\mathcal{I}$*-usr-resta*) (λ(-, *s-rest*). *pred-prod I-key-rest I-auth-rest s-rest*) *ideal-rest′*
⟨*proof*⟩

lemma *WT-real-core′* [*WT-intro*]: *WT-core* $\mathcal{I}$*-adv-core* $\mathcal{I}$*-usr-core invar-real′ real-core′*
⟨*proof*⟩ **lemma** *trace-eq-sec*:
fixes $\mathcal{I}$*-adv-restk* $\mathcal{I}$*-adv-resta* $\mathcal{I}$*-usr-restk* $\mathcal{I}$*-usr-resta*
defines *outs-adv* ≡ (*UNIV* <+> *UNIV* <+> *UNIV* <+> *sec.Inp-Fedit* ‘ *carrier* $\mathcal{L}$) <+> *outs-*$\mathcal{I}$ ($\mathcal{I}$*-adv-restk* $\oplus_{\mathcal{I}}$ $\mathcal{I}$*-adv-resta*)
and *outs-usr* ≡ (*sec.Inp-Send* ‘ *carrier* $\mathcal{L}$ <+> *UNIV*) <+> *outs-*$\mathcal{I}$ ($\mathcal{I}$*-usr-restk* $\oplus_{\mathcal{I}}$ $\mathcal{I}$*-usr-resta*)
assumes *WT-key* [*WT-intro*]: *WT-rest* $\mathcal{I}$*-adv-restk* $\mathcal{I}$*-usr-restk I-key-rest key-rest*

and *WT-auth* [*WT-intro*]: *WT-rest* $\mathcal{I}$*-adv-resta* $\mathcal{I}$*-usr-resta I-auth-rest auth-rest*
shows (*outs-adv* <+> *outs-usr*) $\vdash_C$ *fused-resource.fuse ideal-core′ ideal-rest′* ((*ideal-s-core′*, *ideal-s-rest′*)) ≈
fused-resource.fuse real-core′ real-rest′ ((*real-s-core′*, *real-s-rest′*))
⟨*proof*⟩

11.7.3 Simplifying the final resource by moving the interfaces from core to rest

lemma *connect*[*unfolded* $\mathcal{I}$*-adv-core-def* $\mathcal{I}$*-usr-core-def*]:
fixes $\mathcal{I}$*-adv-restk* $\mathcal{I}$*-adv-resta* $\mathcal{I}$*-usr-restk* $\mathcal{I}$*-usr-resta*
defines $\mathcal{I}$ ≡ ($\mathcal{I}$*-adv-core* $\oplus_{\mathcal{I}}$ ($\mathcal{I}$*-adv-restk* $\oplus_{\mathcal{I}}$ $\mathcal{I}$*-adv-resta*)) $\oplus_{\mathcal{I}}$ ($\mathcal{I}$*-usr-core* $\oplus_{\mathcal{I}}$ ($\mathcal{I}$*-usr-restk* $\oplus_{\mathcal{I}}$ $\mathcal{I}$*-usr-resta*))
assumes [*WT-intro*]: *WT-rest* $\mathcal{I}$*-adv-restk* $\mathcal{I}$*-usr-restk I-key-rest key-rest*
and [*WT-intro*]: *WT-rest* $\mathcal{I}$*-adv-resta* $\mathcal{I}$*-usr-resta I-auth-rest auth-rest*
and *exception-*$\mathcal{I}$ $\mathcal{I}$ ⊢*g D* √
shows *connect D* (*obsf-resource ideal-resource*) = *connect D* (*obsf-resource real-resource*)
⟨*proof*⟩

end

end

end

11.8 Concrete security

context *one-time-pad* **begin**

lemma *WT-enc-callee* [*WT-intro*]:
$\mathcal{I}$-uniform (*sec.Inp-Send* ‘ *carrier* $\mathcal{L}$) *UNIV*, *$\mathcal{I}$-uniform UNIV* (*key.Out-Alice* ‘ *carrier* $\mathcal{L}$) $\oplus_{\mathcal{I}}$ *$\mathcal{I}$-uniform* (*sec.Inp-Send* ‘ *carrier* $\mathcal{L}$) *UNIV* $\vdash_C$ *CNV enc-callee* ()
$\surd$
⟨*proof*⟩

lemma *WT-dec-callee* [*WT-intro*]:
$\mathcal{I}$-uniform UNIV (*sec.Out-Recv* ‘ *carrier* $\mathcal{L}$), *$\mathcal{I}$-uniform UNIV* (*key.Out-Bob* ‘ *carrier* $\mathcal{L}$) $\oplus_{\mathcal{I}}$ *$\mathcal{I}$-uniform UNIV* (*sec.Out-Recv* ‘ *carrier* $\mathcal{L}$) $\vdash_C$ *CNV dec-callee* ()
$\surd$
⟨*proof*⟩

lemma *pfinite-enc-callee* [*pfinite-intro*]:
pfinite-converter (*$\mathcal{I}$-uniform* (*sec.Inp-Send* ‘ *carrier* $\mathcal{L}$) *UNIV*) (*$\mathcal{I}$-uniform UNIV* (*key.Out-Alice* ‘ *carrier* $\mathcal{L}$) $\oplus_{\mathcal{I}}$ *$\mathcal{I}$-uniform* (*sec.Inp-Send* ‘ *carrier* $\mathcal{L}$) *UNIV*) (*CNV enc-callee* ())
⟨*proof*⟩

lemma *pfinite-dec-callee* [*pfinite-intro*]:
pfinite-converter (*$\mathcal{I}$-uniform UNIV* (*sec.Out-Recv* ‘ *carrier* $\mathcal{L}$)) (*$\mathcal{I}$-uniform UNIV* (*key.Out-Bob* ‘ *carrier* $\mathcal{L}$) $\oplus_{\mathcal{I}}$ *$\mathcal{I}$-uniform UNIV* (*sec.Out-Recv* ‘ *carrier* $\mathcal{L}$)) (*CNV dec-callee* ())
⟨*proof*⟩

context
fixes
key-rest :: (*'key-s-rest*, *key.event*, *'key-iadv-rest*, *'key-iusr-rest*, *'key-oadv-rest*, *'key-ousr-rest*) *rest-wstate* **and**
auth-rest :: (*'auth-s-rest*, *auth.event*, *'auth-iadv-rest*, *'auth-iusr-rest*, *'auth-oadv-rest*, *'auth-ousr-rest*) *rest-wstate* **and**
$\mathcal{I}$-adv-restk **and** *$\mathcal{I}$-adv-resta* **and** *$\mathcal{I}$-usr-restk* **and** *$\mathcal{I}$-usr-resta* **and** *I-key-rest* **and** *I-auth-rest*
assumes
WT-key-rest [*WT-intro*]: *WT-rest $\mathcal{I}$-adv-restk $\mathcal{I}$-usr-restk I-key-rest key-rest* **and**
WT-auth-rest [*WT-intro*]: *WT-rest $\mathcal{I}$-adv-resta $\mathcal{I}$-usr-resta I-auth-rest auth-rest*
begin

theorem *secure*:
defines *$\mathcal{I}$-real* ≡ ((*$\mathcal{I}$-full* $\oplus_{\mathcal{I}}$ (*$\mathcal{I}$-full* $\oplus_{\mathcal{I}}$ (*$\mathcal{I}$-full* $\oplus_{\mathcal{I}}$ *$\mathcal{I}$-uniform* (*sec.Inp-Fedit* ‘ (*carrier* $\mathcal{L}$)) *UNIV*))) $\oplus_{\mathcal{I}}$ (*$\mathcal{I}$-adv-restk* $\oplus_{\mathcal{I}}$ *$\mathcal{I}$-adv-resta*))
and *$\mathcal{I}$-common-core* ≡ *$\mathcal{I}$-uniform* (*sec.Inp-Send* ‘ (*carrier* $\mathcal{L}$)) *UNIV* $\oplus_{\mathcal{I}}$ *$\mathcal{I}$-uniform UNIV* (*sec.Out-Recv* ‘ *carrier* $\mathcal{L}$)
and *$\mathcal{I}$-common-rest* ≡ *$\mathcal{I}$-usr-restk* $\oplus_{\mathcal{I}}$ *$\mathcal{I}$-usr-resta*
and *$\mathcal{I}$-ideal* ≡ (*$\mathcal{I}$-full* $\oplus_{\mathcal{I}}$ (*$\mathcal{I}$-full* $\oplus_{\mathcal{I}}$ *$\mathcal{I}$-uniform* (*sec.Inp-Fedit* ‘ (*carrier* $\mathcal{L}$)) *UNIV*)) $\oplus_{\mathcal{I}}$ (*$\mathcal{I}$-adv-restk* $\oplus_{\mathcal{I}}$ *$\mathcal{I}$-adv-resta*)
shows *constructive-security-obsf* (*real-resource TYPE*(-) *TYPE*(-) *key-rest auth-rest*) (*sec.resource* (*ideal-rest key-rest auth-rest*)) (*sim* $|_=$ *1$_C$*) *$\mathcal{I}$-real $\mathcal{I}$-ideal* (*$\mathcal{I}$-common-core* $\oplus_{\mathcal{I}}$ *$\mathcal{I}$-common-rest*) $\mathcal{A}$ *0*

⟨*proof*⟩

end

end

11.9 Asymptotic security

locale *one-time-pad′* =
  fixes $\mathcal{L}$ :: *security* ⇒ (*'msg*, *'more*) *boolean-algebra-scheme*
  assumes *one-time-pad* [*locale-witness*]: $\bigwedge\eta$. *one-time-pad* ($\mathcal{L}$ η)
begin

sublocale *one-time-pad* $\mathcal{L}$ η **for** η ⟨*proof*⟩

definition *real-resource′* **where** *real-resource′ rest1 rest2 η* = *real-resource TYPE*(-) *TYPE*(-) *η* (*rest1 η*) (*rest2 η*)
definition *ideal-resource′* **where** *ideal-resource′ rest1 rest2 η* = *sec.resource η* (*ideal-rest* (*rest1 η*) (*rest2 η*))
definition *sim′* **where** *sim′ η* = (*sim* $|_=$ 1_C)

context
  fixes
   key-rest :: *nat* ⇒ (*'key-s-rest*, *key.event*, *'key-iadv-rest*, *'key-iusr-rest*, *'key-oadv-rest*, *'key-ousr-rest*) *rest-wstate* **and**
    auth-rest :: *nat* ⇒ (*'auth-s-rest*, *auth.event*, *'auth-iadv-rest*, *'auth-iusr-rest*, *'auth-oadv-rest*, *'auth-ousr-rest*) *rest-wstate* **and**
    $\mathcal{I}$*-adv-restk* **and** $\mathcal{I}$*-adv-resta* **and** $\mathcal{I}$*-usr-restk* **and** $\mathcal{I}$*-usr-resta* **and** *I-key-rest* **and** *I-auth-rest*
  assumes
   WT-key-res: $\bigwedge\eta$. *WT-rest* ($\mathcal{I}$*-adv-restk η*) ($\mathcal{I}$*-usr-restk η*) (*I-key-rest η*) (*key-rest η*) **and**
    WT-auth-rest: $\bigwedge\eta$. *WT-rest* ($\mathcal{I}$*-adv-resta η*) ($\mathcal{I}$*-usr-resta η*) (*I-auth-rest η*) (*auth-rest η*)
begin

theorem *secure′*:
  defines $\mathcal{I}$*-real* ≡ $\lambda\eta$. (($\mathcal{I}$*-full* $\oplus_{\mathcal{I}}$ ($\mathcal{I}$*-full* $\oplus_{\mathcal{I}}$ ($\mathcal{I}$*-full* $\oplus_{\mathcal{I}}$ $\mathcal{I}$*-uniform* (*sec.Inp-Fedit* ‘ (*carrier* ($\mathcal{L}$ η))) *UNIV*))) $\oplus_{\mathcal{I}}$ ($\mathcal{I}$*-adv-restk η* $\oplus_{\mathcal{I}}$ $\mathcal{I}$*-adv-resta η*))
    and $\mathcal{I}$*-common* ≡ $\lambda\eta$. (($\mathcal{I}$*-uniform* (*sec.Inp-Send* ‘ (*carrier* ($\mathcal{L}$ η))) *UNIV* $\oplus_{\mathcal{I}}$ $\mathcal{I}$*-uniform UNIV* (*sec.Out-Recv* ‘ *carrier* ($\mathcal{L}$ η))) $\oplus_{\mathcal{I}}$ ($\mathcal{I}$*-usr-restk η* $\oplus_{\mathcal{I}}$ $\mathcal{I}$*-usr-resta η*))
    and $\mathcal{I}$*-ideal* ≡ $\lambda\eta$. ($\mathcal{I}$*-full* $\oplus_{\mathcal{I}}$ ($\mathcal{I}$*-full* $\oplus_{\mathcal{I}}$ $\mathcal{I}$*-uniform* (*sec.Inp-Fedit* ‘ (*carrier* ($\mathcal{L}$ η))) *UNIV*)) $\oplus_{\mathcal{I}}$ ($\mathcal{I}$*-adv-restk η* $\oplus_{\mathcal{I}}$ $\mathcal{I}$*-adv-resta η*)
  shows *constructive-security-obsf′* (*real-resource′ key-rest auth-rest*) (*ideal-resource′ key-rest auth-rest*) *sim′* $\mathcal{I}$*-real* $\mathcal{I}$*-ideal* $\mathcal{I}$*-common* $\mathcal{A}$
⟨*proof*⟩

end

end

end
theory *Diffie-Hellman-CC*
imports
Game-Based-Crypto.Diffie-Hellman
../Asymptotic-Security
../Construction-Utility
../Specifications/Key
../Specifications/Channel
begin

hide-const (**open**) *Resumption.Pause Monomorphic-Monad.Pause Monomorphic-Monad.Done*

no-notation *Sublist.parallel* (**infixl** ‖ *50*)
no-notation *plus-oracle* (**infix** $\oplus_O$ *500*)

12 Diffie-Hellman construction

locale *diffie-hellman* =
auth: *ideal-channel id* :: *'grp* ⇒ *'grp False* +
key: *ideal-key carrier* $\mathcal{G}$ +
cyclic-group $\mathcal{G}$
for
$\mathcal{G}$ :: *'grp cyclic-group* (**structure**)
begin

12.1 Defining user callees

datatype *'grp' cstate* = *CState-Void* | *CState-Half nat* | *CState-Full nat* × *'grp'*

datatype *icnv-alice* = *Inp-Activation-Alice*
datatype *icnv-bob* = *Iact-Activation-Bob*

datatype *ocnv-alice* = *Out-Activation-Alice*
datatype *ocnv-bob* = *Out-Activation-Bob*

fun *alice-callee* :: *'grp cstate* ⇒ *key.iusr-alice* + *icnv-alice*
⇒ ((*'grp key.ousr-alice* + *ocnv-alice*) × *'grp cstate*, *'grp auth.iusr-alice* + *auth.iusr-bob*,
auth.ousr-alice + *'grp auth.ousr-bob*) *gpv*
where
alice-callee CState-Void (*Inr* -) = *do* {
x ← *lift-spmf* (*sample-uniform* (*order* $\mathcal{G}$));
let msg = **g** [^] *x*;
Pause
(*Inl* (*auth.Inp-Send msg*))
(λ*rsp*. *case rsp of*

```
        Inl - ⇒ Done (Inr Out-Activation-Alice, CState-Half x)
      | Inr - ⇒ Fail) }
  | alice-callee (CState-Half x) (Inl -) =
      Pause
        (Inr auth.Inp-Recv)
        (λrsp. case rsp of
          Inl - ⇒ Fail
        | Inr msg ⇒ case msg of
            auth.Out-Recv gy ⇒
              let key = gy [^] x in
              Done (Inl (key.Out-Alice key), CState-Full (x, key)))
  | alice-callee (CState-Full (x, key)) (Inl -) = Done (Inl (key.Out-Alice key),
CState-Full (x, key))
  | alice-callee - - = Fail

fun bob-callee :: 'grp cstate ⇒ key.iusr-bob + icnv-bob
 ⇒ (('grp key.ousr-bob + ocnv-bob) × 'grp cstate, auth.iusr-bob + 'grp auth.iusr-alice,
'grp auth.ousr-bob + auth.ousr-alice) gpv
  where
    bob-callee CState-Void (Inr -) = do {
      y ← lift-spmf (sample-uniform (order G));
      let msg = g [^] y;
      Pause
        (Inr (auth.Inp-Send msg))
        (λrsp. case rsp of
          Inl - ⇒ Fail
        | Inr - ⇒ Done (Inr Out-Activation-Bob, CState-Half y)) }
  | bob-callee (CState-Half y) (Inl -) =
      Pause
        (Inl auth.Inp-Recv)
        (λrsp. case rsp of
          Inl msg ⇒ case msg of
            auth.Out-Recv gx ⇒
              let k = gx [^] y in
              Done (Inl (key.Out-Bob k), CState-Full (y, k))
        | Inr - ⇒ Fail)
  | bob-callee (CState-Full (y, key)) (Inl -) =  Done (Inl (key.Out-Bob key),
CState-Full (y, key))
  | bob-callee - - = Fail
```

12.2 Defining adversary callee

```
type-synonym 'grp' astate = ('grp' × 'grp') option

type-synonym 'grp' isim = 'grp' auth.iadv + 'grp' auth.iadv
datatype osim = Out-Simulator

fun sim-callee-base :: (('grp × 'grp) ⇒ 'grp ) ⇒ ('grp astate, 'grp auth.iadv, 'grp
auth.oadv) oracle'
```

where
sim-callee-base - - (Inl -) = return-pmf None
| *sim-callee-base pick gpair-opt (Inr (Inl -)) = do {*
sample ← do {
x ← sample-uniform (order 𝒢);
y ← sample-uniform (order 𝒢);
*return-spmf (**g** [^] x, **g** [^] y) };*
let sample′ = case-option sample id gpair-opt;
return-spmf (Inr (Inl (auth.Out-Look (pick sample′))), Some sample′) }
| *sim-callee-base - gpair-opt (Inr (Inr -)) = return-spmf (Inr (Inr auth.Out-Fedit), gpair-opt)*

fun *sim-callee* :: *′grp astate* ⇒ *′grp auth.iadv + ′grp auth.iadv*
⇒ ((*′grp auth.oadv + ′grp auth.oadv*) × *′grp astate*, *key.iadv + ′grp isim*, *key.oadv + osim*) *gpv*
where
sim-callee s-gpair query =
(*let handle* = (*λgpair-pick wrap-out q-split. do {*
- ← Pause (Inr query) Done;
(*out*, *s-gpair′*) ← *lift-spmf* (*sim-callee-base gpair-pick s-gpair q-split*);
Done (*wrap-out out*, *s-gpair′*) }) *in*
case-sum (*handle fst Inl*) (*handle snd Inr*) *query*)

12.3 Defining event-translator

datatype *estate-base = EState-Void | EState-Store | EState-Collect*
type-synonym *estate = bool* × (*estate-base* × *auth.s-shell*) × *estate-base* × *auth.s-shell*

definition *einit* :: *estate*
where
einit ≡ (*False*, (*EState-Void*, {}), *EState-Void*, {})

definition *eleak* :: (*estate*, *key.event*, *′grp isim*, *osim*) *eoracle*
where
eleak ≡ (*λ*(*s-flg*, (*s-event1*, *s-shell1*), *s-event2*, *s-shell2*) *query.*
let handle-arg1 = (*λs q. case* (*s*, *q*) *of* (*EState-Store*, *Some* (*Inr* (*Inr -*))) ⇒ (*True*, *EState-Collect*) | (*s′*, -) ⇒ (*False*, *s′*)) *in*
let handle-arg2 = (*λs q D. case* (*s*, *q*) *of* (*EState-Store*, *Inr* -) ⇒ *D* | - ⇒ *return-pmf None*) *in*
let (*is-ch1*, *s-event1′*) = *handle-arg1 s-event1* (*case-sum Some* (*λ-. None*) *query*) *in*
let (*is-ch2*, *s-event2′*) = *handle-arg1 s-event2* (*case-sum* (*λ-. None*) *Some query*) *in*
let check-pst1 = *is-ch1* ∧ *s-event2′* ≠ *EState-Void* ∧ *auth.Bob* ∈ *s-shell1* ∧ *auth.Alice* ∈ *s-shell2 in*
let check-pst2 = *is-ch2* ∧ *s-event1′* ≠ *EState-Void* ∧ *auth.Alice* ∈ *s-shell1* ∧ *auth.Bob* ∈ *s-shell2 in*
let e-pstfix1 = *if check-pst1 then* [*key.Event-Shell key.Bob*] *else* [] *in*

```
    let e-pstfix2 = if check-pst2 then [key.Event-Shell key.Alice] else [] in
    let e-prefix = if ¬s-flg then [key.Event-Kernel] else [] in
      let (s-flg', event) = if is-ch2 ∨ is-ch1 then (True, e-prefix @ e-pstfix1 @
e-pstfix2) else (s-flg, []) in
       let result-base = return-spmf ((Out-Simulator, event), s-flg', (s-event1',
s-shell1), s-event2', s-shell2) in
    case-sum (handle-arg2 s-event1) (handle-arg2 s-event2) query result-base)

fun etran-base :: (key.party × key.party ⇒ key.party × key.party)
  ⇒ (estate, auth.event, key.event list) oracle'
  where
   etran-base mod-event (s-flg, (s-event1, s-shell1), s-event2, s-shell2) (auth.Event-Shell
party) =
    (let party-dual = auth.case-party (auth.Bob) (auth.Alice) party in
    let epair = auth.case-party prod.swap id party (key.Bob, key.Alice) in
     let (s-event-eq, s-event-neq) = auth.case-party prod.swap id party (s-event1,
s-event2) in
    let check = party-dual ∈ s-shell2 ∧ s-event-eq = EState-Collect ∧ s-event-neq
≠ EState-Void in
    let event = if check then [key.Event-Shell ((fst o mod-event) epair)] else [] in
    let s-shell1' = insert party s-shell1 in
    if party ∈ s-shell1 then
      return-pmf None
    else
      return-spmf (event, s-flg, (s-event1, s-shell1'), s-event2, s-shell2))

fun etran :: (estate, (icnv-alice + icnv-bob) + auth.event + auth.event, key.event
list) oracle'
  where
    etran (s-flg, (EState-Void, s-shell1), s-event2, s-shell2) (Inl (Inl -)) =
     (let check = (s-event2 = EState-Collect ∧ auth.Alice ∈ s-shell1 ∧ auth.Bob ∈
s-shell2) in
     let event = if check then [key.Event-Shell key.Alice] else [] in
     let state = (s-flg, (EState-Store, s-shell1), s-event2, s-shell2) in
     if auth.Alice ∈ s-shell1 then return-spmf (event, state) else return-pmf None)
  | etran (s-flg, (s-event1, s-shell1), EState-Void, s-shell2) (Inl (Inr -)) =
     (let check = (s-event1 = EState-Collect ∧ auth.Bob ∈ s-shell1 ∧ auth.Alice ∈
s-shell2) in
     let event = if check then [key.Event-Shell key.Bob] else [] in
     let state =  (s-flg, (s-event1, s-shell1), EState-Store, s-shell2) in
     if auth.Alice ∈ s-shell2 then return-spmf (event, state) else return-pmf None)
  | etran state (Inr query) =
     (let handle = (λmod-s mod-e q. do {
       (evt, state') ← etran-base mod-e (mod-s state) q;
       return-spmf (evt, mod-s state')  }) in
     case-sum (handle id id) (handle (apsnd prod.swap) prod.swap) query)
  | etran - - = return-pmf None
```

12.4 Defining Ideal and Real constructions

context
fixes
auth1-rest :: (*'auth1-s-rest*, *auth.event*, *'auth1-iadv-rest*, *'auth1-iusr-rest*, *'auth1-oadv-rest*, *'auth1-ousr-rest*) *rest-wstate* **and**
auth2-rest :: (*'auth2-s-rest*, *auth.event*, *'auth2-iadv-rest*, *'auth2-iusr-rest*, *'auth2-oadv-rest*, *'auth2-ousr-rest*) *rest-wstate*
begin

primcorec *ideal-core-alt*
where
cpoke ideal-core-alt = *cpoke* (*translate-core etran key.core*)
| *cfunc-adv ideal-core-alt* = †(*cfunc-adv key.core*) ⊕$_O$ (λ(*se*, *sc*) *q*. *do* {
((*out*, *es*), *se'*) ← *eleak se q*;
sc' ← *foldl-spmf* (*cpoke key.core*) (*return-spmf sc*) *es*;
return-spmf (*out*, *se'*, *sc'*) })
| *cfunc-usr ideal-core-alt* = *cfunc-usr* (*translate-core etran key.core*)

primcorec *ideal-rest-alt*
where
rinit ideal-rest-alt = *rinit* (*parallel-rest auth1-rest auth2-rest*)
| *rfunc-adv ideal-rest-alt* = (λ*s q*. *map-spmf* (*apfst* (*apsnd* (*map Inr*))) (*rfunc-adv* (*parallel-rest auth1-rest auth2-rest*) *s q*))
| *rfunc-usr ideal-rest-alt* = (
let handle = *map-sum* (λ- :: *icnv-alice*. *Out-Activation-Alice*) (λ- :: *icnv-bob*. *Out-Activation-Bob*) *in*
plus-eoracle (λ*s q*. *return-spmf* ((*handle q*, [*q*]), *s*)) (*rfunc-usr* (*parallel-rest auth1-rest auth2-rest*)))

primcorec *ideal-rest*
where
rinit ideal-rest = (*einit*, *rinit ideal-rest-alt*)
| *rfunc-adv ideal-rest* = (λ*s q*. *case q of*
Inl ql ⇒ *map-spmf* (*apfst* (*map-prod Inl id*)) (*eleak*† *s ql*)
| *Inr qr* ⇒ *map-spmf* (*apfst* (*map-prod Inr id*)) (*translate-eoracle etran* †(*rfunc-adv ideal-rest-alt*) *s qr*))
| *rfunc-usr ideal-rest* = *translate-eoracle etran* †(*rfunc-usr ideal-rest-alt*)

definition *ideal-resource*
where
ideal-resource ≡
(*let sim* = *CNV sim-callee None in*
attach ((*sim* |$_=$ 1_C) ⊙ *lassocr*$_C$ |$_=$ 1_C |$_=$ 1_C) (*key.resource ideal-rest*))

definition *real-resource*
where
real-resource ≡
(*let dh-wiring* = *parallel-wiring* ⊙ (*CNV alice-callee CState-Void* |$_=$ *CNV bob-callee CState-Void*) ⊙ *parallel-wiring* ⊙ (1_C |$_=$ *swap*$_C$) *in*

attach (((*1*$_C$ $|_=$ *1*$_C$) $|_=$ *rassocl*$_C$ $\odot$ (*dh-wiring* $|_=$ *1*$_C$)) $\odot$ *fused-wiring*) ((*auth.resource auth1-rest*) $\|$ (*auth.resource auth2-rest*)))

12.5 Wiring and simplifying the Ideal construction

abbreviation *basic-rest-sinit*
  where
    basic-rest-sinit ≡ ((), ()), *rinit auth1-rest*, *rinit auth2-rest*)

primcorec *basic-rest* :: ((*unit* × *unit*) × -, -, -, -, -, -, -) *rest-scheme*
  where
    rinit basic-rest = (*rinit auth1-rest*, *rinit auth2-rest*)
  | *rfunc-adv basic-rest* = †(*parallel-eoracle* (*rfunc-adv auth1-rest*) (*rfunc-adv auth2-rest*))
  | *rfunc-usr basic-rest* = †(*parallel-eoracle* (*rfunc-usr auth1-rest*) (*rfunc-usr auth2-rest*))

definition *ideal-s-core′* :: (′*grp astate* × -) × - × ′*grp key.state*
  where
    ideal-s-core′ ≡ ((*None*, ()), *einit*, *key.PState-Store*, {})

definition *ideal-s-rest′*
  where
    ideal-s-rest′ ≡ *basic-rest-sinit*

primcorec *ideal-core′*
  where
    cpoke ideal-core′ = (λ(*s-cnv*, *s-event*, *s-core*) *event*. *do* {
      (*events*, *s-event′*) ← (*etran s-event event*);
      s-core′ ← *foldl-spmf key.poke* (*return-spmf s-core*) *events*;
      return-spmf (*s-cnv*, *s-event′*, *s-core′*) })
  | *cfunc-adv ideal-core′* = (λ((*s-sim*, -), *s-event-core*) *q*.
      map-spmf
        (λ((*out*, *s-sim′*), *s-event-core′*). (*out*, (*s-sim′*, ()), *s-event-core′*))
        (*exec-gpv*
          (†*key.iface-adv* ⊕$_O$ (λ(*se*, *sc*) *isim*. *do* {
            ((*out*, *es*), *se′*) ← *eleak se isim*;
            sc′ ← *foldl-spmf* (*cpoke key.core*) (*return-spmf sc*) *es*;
            return-spmf (*out*, *se′*, *sc′*) }))
          (*sim-callee s-sim q*) *s-event-core*))
  | *cfunc-usr ideal-core′* = (λ(*s-cnv*, *s-core*) *q*.
      map-spmf (λ(*out*, *s-core′*). (*out*, *s-cnv*, *s-core′*)) (†*key.iface-usr s-core q*))

primcorec *ideal-rest′*
  where
    rinit ideal-rest′ = *rinit basic-rest*
  | *rfunc-adv ideal-rest′* = (λ*s q*. *map-spmf* (*apfst* (*apsnd* (*map Inr*))) (*rfunc-adv basic-rest s q*))
  | *rfunc-usr ideal-rest′* = (
      let handle = *map-sum* (λ- :: *icnv-alice*. *Out-Activation-Alice*) (λ- :: *icnv-bob*. *Out-Activation-Bob*) *in*

plus-eoracle ($\lambda s\ q.$ *return-spmf* ((*handle q*, [*q*]), *s*)) (*rfunc-usr basic-rest*))

12.5.1 The ideal attachment lemma

context
begin

lemma *ideal-resource-shift-interface*: *key.resource ideal-rest* = *RES*
(*apply-wiring* (*rassocl*$_w$ |$_w$ (*id*, *id*)) (*fused-resource.fuse ideal-core-alt ideal-rest-alt*))

((*einit*, *key.PState-Store*, {}), *rinit ideal-rest-alt*)
⟨*proof*⟩ **lemma** *ideal-resource-alt-def*: *ideal-resource* =
(*let sim* = *CNV sim-callee None in*
let s-init = ((*einit*, *key.PState-Store*, {}), *rinit ideal-rest-alt*) *in*
attach ((*sim* |$_=$ *1*$_C$) |$_=$ *1*$_C$ |$_=$ *1*$_C$) (*RES* (*fused-resource.fuse ideal-core-alt ideal-rest-alt*) *s-init*))
⟨*proof*⟩

lemma *attach-ideal*: *ideal-resource* = *RES* (*fused-resource.fuse ideal-core′ ideal-rest′*) (*ideal-s-core′*, *ideal-s-rest′*)
⟨*proof*⟩

end

12.6 Wiring and simplifying the Real construction

definition *real-s-core′* :: (- × *′grp cstate* × *′grp cstate*) × *′grp auth.state* × *′grp auth.state*
where
real-s-core′ ≡ ((), *CState-Void*, *CState-Void*), (*auth.State-Void*, {}), (*auth.State-Void*, {}))

definition *real-s-rest′*
where
real-s-rest′ ≡ *basic-rest-sinit*

primcorec *real-core′* :: ((*unit* × -) × -, -, -, -, -, -) *core*
where
cpoke real-core′ = (λ(*s-advusr*, *s-core*) *event*.
map-spmf (*Pair s-advusr*) (*parallel-handler auth.poke auth.poke s-core event*))
| *cfunc-adv real-core′* = †(*auth.iface-adv* ‡$_O$ *auth.iface-adv*)
| *cfunc-usr real-core′* = (λ((*s-adv*, *s-usr*), *s-core*) *iusr*.
let handle-req = *lsumr* ∘ *map-sum id* (*rsuml* ∘ *map-sum swap-sum id* ∘ *lsumr*) ∘ *rsuml in*
let handle-ret = *lsumr* ∘ (*map-sum id* (*rsuml* ∘ (*map-sum swap-sum id* ∘ *lsumr*)) ∘ *rsuml*) ∘ *map-sum id swap-sum in*
let handle-inp = *map-sum id swap-sum* ∘ (*lsumr* ∘ *map-sum id* (*rsuml* ∘ *map-sum swap-sum id* ∘ *lsumr*) ∘ *rsuml*) *in*
let handle-out = *apfst* (*lsumr* ∘ (*map-sum id* (*rsuml* ∘ (*map-sum swap-sum id* ∘ *lsumr*)) ∘ *rsuml*)) *in*

map-spmf
(λ((*ousr*, *s-usr′*), *s-core′*). (*ousr*, (*s-adv*, *s-usr′*), *s-core′*))
(*exec-gpv*
(*auth.iface-usr* ‡$_O$ *auth.iface-usr*)
(*map-gpv′*
handle-out handle-inp handle-ret
((*alice-callee* ‡$_I$ *bob-callee*) *s-usr* (*handle-req iusr*)))
s-core))

definition *real-rest′* :: ((*unit* × *unit*) × -, -, -, -, -, -, -) *rest-scheme*
where
real-rest′ ≡ *basic-rest*

12.6.1 The real attachment lemma

lemma *attach-real*: *real-resource* = *1*$_C$ |$_=$ *rassocl*$_C$ ▷ *RES* (*fused-resource.fuse real-core′ real-rest′*) (*real-s-core′*, *real-s-rest′*)
⟨*proof*⟩

12.7 A lazy construction and its DH reduction

12.7.1 Defining a lazy construction with an inlined sampler

type-synonym *′grp′ st-state* = (*′grp′* × *′grp′* × *′grp′*) *option*
type-synonym *′grp′ bc-state* = (*′grp′ st-state* × *′grp′ cstate* × *′grp′ cstate*) × *′grp′ auth.state* × *′grp′ auth.state*

context
fixes *sample-triple* :: (*′grp* × *′grp* × *′grp*) *spmf*
begin

abbreviation *basic-core-sinit* :: *′grp bc-state*
where
basic-core-sinit ≡ ((*None*, *CState-Void*, *CState-Void*), (*auth.State-Void*, {}), *auth.State-Void*, {})

fun *basic-core-helper-base* :: (*′grp bc-state*, *unit*, *unit*) *oracle′*
where
basic-core-helper-base ((*s-key*, *CState-Void*, *s-cnv2*), (*auth.State-Void*, *parties1*), *s-auth2*) - =
(*if auth.Alice* ∈ *parties1*
then return-spmf ((), (*s-key*, *CState-Half 0*, *s-cnv2*), (*auth.State-Store* **1**, *parties1*), *s-auth2*)
else return-pmf None)
| *basic-core-helper-base* - - = *return-pmf None*

definition *basic-core-helper* :: (*′grp bc-state*, *icnv-alice* + *icnv-bob*) *handler*
where
basic-core-helper ≡ (λ*state query*.
let handle = λ((*sk*, (*sc1*, *sc2*)), *sa1*, *sa2*). ((*sk*, (*sc2*, *sc1*)), *sa2*, *sa1*) *in*

```
    let func = λh-s f s. map-spmf (h-s o snd) (f (h-s s) ()) in
    let func-alc = func id basic-core-helper-base in
    let func-bob = func handle basic-core-helper-base in
    case-sum (λ-. func-alc state) (λ-. func-bob state) query)

fun basic-core-oracle-adv :: unit + unit ⇒ ('grp st-state × 'grp auth.state, 'grp
auth.iadv, 'grp auth.oadv) oracle'
  where
   basic-core-oracle-adv sel (None, auth.State-Store -, parties) (Inr (Inl -)) = do {
     (gxy, gx, gy) ← sample-triple;
     let out = case-sum (λ-. gx) (λ-. gy) sel;
    return-spmf (Inr (Inl (auth.Out-Look out)), Some (gxy, gx, gy), auth.State-Store
1, parties)
    }
  | basic-core-oracle-adv sel (Some dhs, auth.State-Store -, parties) (Inr (Inl -)) =
     (case dhs of (gxy, gx, gy) ⇒
       let out = case-sum (λ-. gx) (λ-. gy) sel in
        return-spmf (Inr (Inl (auth.Out-Look out)), Some dhs, auth.State-Store 1,
parties))
  | basic-core-oracle-adv - (s-key, auth.State-Store -, parties) (Inr (Inr -)) =
     return-spmf (Inr (Inr auth.Out-Fedit), s-key, auth.State-Collect 1, parties)
  | basic-core-oracle-adv - - - = return-pmf None


fun basic-core-oracle-usr-base :: ('grp bc-state, unit, 'grp) oracle'
  where
   basic-core-oracle-usr-base ((s-key, CState-Half -, s-cnv2), s-auth1, auth.State-Collect
-, parties2) - =
    (let h-state = λk. ((Some k, CState-Full (0, 1), s-cnv2), s-auth1, auth.State-Collected,
parties2) in
     if auth.Bob ∈ parties2 then
      (case s-key of
        None ⇒ do {
          (gxy, gx, gy) ← sample-triple;
          return-spmf (gxy, h-state (gxy, gx, gy)) }
       | Some (gxy, gx, gy) ⇒ return-spmf (gxy, h-state (gxy, gx, gy)))
     else return-pmf None)
 | basic-core-oracle-usr-base ((Some dhs, CState-Full -, s-cnv2), s-auth1, auth.State-Collected,
parties2) - =
     (case dhs of (gxy, gx, gy) ⇒
    return-spmf (gxy, (Some dhs, CState-Full (0, 1), s-cnv2), s-auth1, auth.State-Collected,
parties2))
  | basic-core-oracle-usr-base - - = return-pmf None

definition basic-core-oracle-usr :: (-, key.iusr-alice + key.iusr-bob, -) oracle'
  where
   basic-core-oracle-usr ≡ (λstate query.
     let handle = λ((sk, (sc1, sc2)), sa1, sa2). ((sk, (sc2, sc1)), sa2, sa1) in
     let func = λh-o h-s f s. map-spmf (map-prod h-o h-s) (f (h-s s) ()) in
```

let func-alc = func (Inl o key.Out-Alice) id basic-core-oracle-usr-base in
let func-bob = func (Inr o key.Out-Bob) handle basic-core-oracle-usr-base in
case-sum (λ-. func-alc state) (λ-. func-bob state) query)

primcorec *basic-core*
where
cpoke basic-core = (λ(s-other, s-core) event.
map-spmf (Pair s-other) (parallel-handler auth.poke auth.poke s-core event))
| *cfunc-adv basic-core = (λ((s-key, s-cnv), s-auth1, s-auth2) iadv.*
let handle = (λsel s-init h-out h-state query.
map-spmf
(λ(out, (s-key′, s-auth′)). (h-out out, (s-key′, s-cnv), h-state s-auth′ s-auth1 s-auth2))
(basic-core-oracle-adv sel (s-key, s-init) query)) in
case-sum (handle (Inl ()) s-auth1 Inl (λx y z. (x, z))) (handle (Inr ()) s-auth2 Inr (λx y z. (y, x))) iadv)
| *cfunc-usr basic-core =*
(let handle = map-sum (λ-. Out-Activation-Alice) (λ-. Out-Activation-Bob) in
basic-core-oracle-usr ⊕O (λs q. map-spmf (Pair (handle q)) (basic-core-helper s q)))

primcorec *lazy-core*
where
cpoke lazy-core = (λs. case-sum (λq. basic-core-helper s q) (cpoke basic-core s))
| *cfunc-adv lazy-core = cfunc-adv basic-core*
| *cfunc-usr lazy-core = basic-core-oracle-usr*

definition *lazy-rest*
where
lazy-rest ≡ ideal-rest′

end

12.7.2 Defining a lazy construction with an external sampler

context
begin

private type-synonym *('grp′, 'iadv-rest′, 'iusr-rest′) dh-inp =*
(('grp′ auth.iadv + 'grp′ auth.iadv) + 'iadv-rest′) + (key.iusr-alice + key.iusr-bob) + (icnv-alice + icnv-bob) + 'iusr-rest′

private type-synonym *('grp′, 'oadv-rest′, 'ousr-rest′) dh-out =*
(('grp′ auth.oadv + 'grp′ auth.oadv) + 'oadv-rest′) + ('grp′ key.ousr-alice + 'grp′ key.ousr-bob) + (ocnv-alice + ocnv-bob) + 'ousr-rest′

fun *interceptor-base-look :: unit + unit ⇒ 'grp st-state × 'grp auth.state*
⇒ ('grp auth.oadv-look × 'grp st-state, unit, 'grp × 'grp × 'grp) gpv
where

```
    interceptor-base-look sel (None, auth.State-Store -, parties) = do {
      (gxy, gx, gy) ← Pause () Done;
      let out = case-sum (λ-. gx) (λ-. gy) sel;
      Done (auth.Out-Look out, Some (gxy, gx, gy))  }
  | interceptor-base-look sel (Some dhs, auth.State-Store -, parties) = (
      case dhs of (gxy, gx, gy) ⇒
      let out = case-sum (λ-. gx) (λ-. gy) sel in
      Done (auth.Out-Look out, Some (gxy, gx, gy)))
  | interceptor-base-look - - = Fail

fun interceptor-base-recv :: 'grp bc-state ⇒ ('grp × 'grp bc-state, unit, 'grp × 'grp
× 'grp) gpv
  where
    interceptor-base-recv ((s-key, CState-Half -, s-cnv2), s-auth1, auth.State-Collect
-, parties2) = (
    let h-state = λk. ((Some k, CState-Full (0, 1), s-cnv2), s-auth1, auth.State-Collected,
parties2) in
      if auth.Bob ∈ parties2 then
        case s-key of
          None ⇒ do {
            (gxy, gx, gy) ← Pause () Done;
            Done (gxy, h-state (gxy, gx, gy))  }
        | Some (gxy, gx, gy) ⇒ Done (gxy, h-state (gxy, gx, gy))
      else
        Fail)
  | interceptor-base-recv ((Some dhs, CState-Full -, s-cnv2), s-auth1, auth.State-Collected,
parties2) = (
      case dhs of (gxy, gx, gy) ⇒
     Done (gxy, (Some dhs, CState-Full (0, 1), s-cnv2), s-auth1, auth.State-Collected,
parties2))
  | interceptor-base-recv - = Fail

fun interceptor ::  - ⇒ (-, -, -) dh-inp  ⇒  (('grp, -, -) dh-out × -, unit, 'grp ×
'grp × 'grp) gpv
  where
    interceptor (sc, sr) (Inl (Inl (q))) = (
       let select-s = (case sc of ((sk, -), sa1, sa2) ⇒ case-sum (λ-. (sk, sa1)) (λ-.
(sk, sa2))) in
      let handle-s = (λx. case sc of ((sk, (sc1, sc2)), sa1, sa2) ⇒ ((x, (sc1, sc2)),
sa1, sa2)) in
      let func-look = (λsel h-o. do {
        (o-look, dhs) ← interceptor-base-look (sel ()) (select-s (sel ())) ;
        Done (Inl (Inl (h-o (Inr (Inl o-look))))), handle-s dhs, sr)  }) in
      let func-dfe = do {
        (out, sc′) ← lift-spmf (cfunc-adv (lazy-core undefined) sc q);
        Done (Inl (Inl out), sc′, sr)  } in
      case q of
        (Inl (Inr (Inl -))) ⇒ func-look Inl Inl
      | (Inr (Inr (Inl -))) ⇒ func-look Inr Inr
```

```
    | - ⇒ func-dfe)
  | interceptor (sc, sr) (Inl (Inr (q))) = do {
      ((out, es), sr′) ← lift-spmf (rfunc-adv lazy-rest sr q);
     sc′ ← lift-spmf (foldl-spmf (λa e. cpoke (lazy-core undefined) a e) (return-spmf
sc) es);
      Done (Inl (Inr out), (sc′, sr′))  }
  | interceptor (sc, sr) (Inr (Inl (q))) = (
      let handle = λ((sk, (sc1, sc2)), sa1, sa2). ((sk, (sc2, sc1)), sa2, sa1) in
      let func-recv = (λh-o h-s. do {
        (o-recv, sc′) ← interceptor-base-recv (h-s sc);
        Done (Inr (Inl (h-o o-recv)), h-s sc′, sr)  }) in
         case-sum (λ-. func-recv (Inl o key.Out-Alice) id) (λ-. func-recv (Inr o
key.Out-Bob) handle) q)
  | interceptor (sc, sr) (Inr (Inr (q))) = do {
      ((out, es), sr′) ← lift-spmf (rfunc-usr lazy-rest sr q);
     sc′ ← lift-spmf (foldl-spmf (λa e. cpoke (lazy-core undefined) a e) (return-spmf
sc) es);
      Done (Inr (Inr out), (sc′, sr′))  }

end
```

12.7.3 Reduction to Diffie-Hellman game

```
definition DH0-sample :: ('grp × 'grp × 'grp) spmf
  where
    DH0-sample = do {
      x ← sample-uniform (order 𝒢);
      y ← sample-uniform (order 𝒢);
      return-spmf ((g [^] x) [^] y, g [^] x, g [^] y)  }

definition DH1-sample :: ('grp × 'grp × 'grp) spmf
  where
    DH1-sample = do {
      x ← sample-uniform (order 𝒢);
      y ← sample-uniform (order 𝒢);
      z ← sample-uniform (order 𝒢);
      return-spmf (g [^] z, g [^] x, g [^] y)  }

lemma lossless-DH0-sample [simp]: lossless-spmf DH0-sample
  ⟨proof⟩

lemma lossless-DH1-sample [simp]: lossless-spmf DH1-sample
  ⟨proof⟩

definition DH-adversary-curry :: ('grp × 'grp × 'grp ⇒ bool spmf) ⇒ 'grp ⇒ 'grp
⇒ 'grp ⇒ bool spmf
  where
    DH-adversary-curry ≡ (λA x y z. bind-spmf (return-spmf (z, x, y)) A)
```

definition *DH-adversary*
where
DH-adversary D ≡ *DH-adversary-curry* (λ*xyz*.
run-gpv (*obsf-oracle* (*obsf-oracle* (λ(*tpl*, *s*) *q*. *map-spmf* (*apsnd* (*Pair tpl*) ∘ *fst*) (*exec-gpv* (λ- -. *return-spmf* (*tpl*, ())) (*interceptor s q*) ()))))
(*obsf-distinguisher D*) (*OK* (*OK* (*xyz*, *basic-core-sinit*, *basic-rest-sinit*))))

context
begin

private abbreviation *S-ic-asm s-cnv1 s-cnv2 s-krn1 s-krn2* ≡
let s-cnvs = {*CState-Void*} ∪ {*CState-Half 0*} ∪ {*CState-Full* (*0*, **1**)} *in*
let s-krns = {*auth.State-Void*} ∪ {*auth.State-Store* **1**} ∪ {*auth.State-Collect* **1**} ∪ {*auth.State-Collected*} *in*
s-cnv1 ∈ *s-cnvs* ∧ *s-cnv2* ∈ *s-cnvs* ∧ *s-krn1* ∈ *s-krns* ∧ *s-krn2* ∈ *s-krns*

private inductive *S-ic* :: ('*grp* × '*grp* × '*grp*) *spmf* ⇒ ('*grp bc-state* × (*unit* × *unit*) × '*auth1-s-rest* × '*auth2-s-rest*) *spmf* ⇒
(('*grp* × '*grp* × '*grp*) × '*grp bc-state* × (*unit* × *unit*) × '*auth1-s-rest* × '*auth2-s-rest*) *spmf* ⇒ *bool*
for *S* :: ('*grp* × '*grp* × '*grp*) *spmf* **where**
S-ic S (*return-spmf* (((*None*, *s-cnv1*, *s-cnv2*), (*s-krn1*, *s-act1*), *s-krn2*, *s-act2*), ((), ()), *s-rest1*, *s-rest2*))
(*map-spmf* (λ*x*. (*x*, (((*None*, *s-cnv1*, *s-cnv2*), (*s-krn1*, *s-act1*), *s-krn2*, *s-act2*), ((), ()), *s-rest1*, *s-rest2*))) *S*)
if *S-ic-asm s-cnv1 s-cnv2 s-krn1 s-krn2*
| *S-ic S* (*return-spmf* (((*Some x*, *s-cnv1*, *s-cnv2*), (*s-krn1*, *s-act1*), *s-krn2*, *s-act2*), ((), ()), *s-rest1*, *s-rest2*))
(*return-spmf* (*x*, (((*Some x*, *s-cnv1*, *s-cnv2*), (*s-krn1*, *s-act1*), *s-krn2*, *s-act2*), ((), ()), *s-rest1*, *s-rest2*)))
if *S-ic-asm s-cnv1 s-cnv2 s-krn1 s-krn2*

private lemma *trace-eq-intercept*:
defines *outs-adv* ≡ ((*UNIV* <+> *UNIV* <+> *UNIV*) <+> *UNIV* <+> *UNIV* <+> *UNIV*) <+> *UNIV* <+> *UNIV*
and *outs-usr* ≡ (*UNIV* <+> *UNIV*) <+> (*UNIV* <+> *UNIV*) <+> *UNIV* <+> *UNIV*
assumes *lossless-spmf sample-triple*
shows *trace-callee-eq* (*fused-resource.fuse* (*lazy-core sample-triple*) *lazy-rest*)
(λ(*tpl*, *s*) *q*. *map-spmf* (*apsnd* (*Pair tpl*) ∘ *fst*) (*exec-gpv* (λ- -. *return-spmf* (*tpl*, ())) (*interceptor s q*) ()))
(*outs-adv* <+> *outs-usr*)
(*return-spmf* (*basic-core-sinit*, *basic-rest-sinit*)) (*pair-spmf sample-triple* (*return-spmf* (*basic-core-sinit*, *basic-rest-sinit*)))
(**is** *trace-callee-eq ?L ?R ?OI ?sl ?sr*)
⟨*proof*⟩ **abbreviation** *dummy x* ≡ *TRY map-spmf OK x ELSE return-spmf Fault*

lemma *reduction*: *advantage D* (*obsf-resource* (*RES* (*fused-resource.fuse* (*lazy-core DH1-sample*) *lazy-rest*) (*basic-core-sinit*, *basic-rest-sinit*)))

(*obsf-resource* (*RES* (*fused-resource.fuse* (*lazy-core DH0-sample*) *lazy-rest*) (*basic-core-sinit*, *basic-rest-sinit*))) = *ddh.advantage* $\mathcal{G}$ (*DH-adversary D*)
⟨*proof*⟩

end

12.8 Proving the trace-equivalence of simplified Ideal and Lazy constructions

context
begin

private abbreviation *isample-nat* ≡ *sample-uniform* (*order* $\mathcal{G}$)
private abbreviation *isample-key* ≡ *spmf-of-set* (*carrier* $\mathcal{G}$)
private abbreviation *isample-pair-nn* ≡ *pair-spmf isample-nat isample-nat*
private abbreviation *isample-pair-nk* ≡ *pair-spmf isample-nat isample-key*

private inductive *S-il* :: ((*'grp astate* × *unit*) × *estate* × *'grp key.state*) *spmf* ⇒ *'grp bc-state spmf* ⇒ *bool*
where
— (Auth1 =a)@(Auth2 =0)
sil-0-0: *S-il* (*return-spmf* ((*None*, ()), (*False*, (*EState-Void*, *s-act1*), *EState-Void*, *s-act2*), *key.PState-Store*, {}))
(*return-spmf* ((*None*, *CState-Void*, *CState-Void*), (*auth.State-Void*, *s-act1*), *auth.State-Void*, *s-act2*))
— ../(Auth1 =a)@(Auth2 =0) # wl
| *sil-1-0*: *S-il* (*return-spmf* ((*None*, ()), (*False*, (*EState-Store*, *s-act1*), *EState-Void*, *s-act2*), *key.PState-Store*, {}))
(*return-spmf* ((*None*, *CState-Half 0*, *CState-Void*), (*auth.State-Store* **1**, *s-act1*), *auth.State-Void*, *s-act2*))
if *auth.Alice* ∈ *s-act1*
| *sil-2-0*: *S-il* (*map-spmf* (λ*k*. ((*None*, ()), (*True*, (*EState-Collect*, *s-act1*), *EState-Void*, *s-act2*), *key.State-Store k*, {})) *isample-key*)
(*return-spmf* ((*None*, *CState-Half 0*, *CState-Void*), (*auth.State-Collect* **1**, *s-act1*), *auth.State-Void*, *s-act2*))
if *auth.Alice* ∈ *s-act1*
— ../(Auth1 =a)@(Auth2 =0) # look
| *sil-1′-0*: *S-il* (*map-spmf* (λ*y*. ((*Some* (**g** [^] *x*, **g** [^] *y*), ()), (*False*, (*EState-Store*, *s-act1*), *EState-Void*, *s-act2*), *key.PState-Store*, {})) *isample-nat*)
(*map-spmf* (λ*yz*. ((*Some* (**g** [^] *snd yz*, **g** [^] (*x* :: *nat*), **g** [^] *fst yz*), *CState-Half 0*, *CState-Void*), (*auth.State-Store* **1**, *s-act1*), *auth.State-Void*, *s-act2*)) *isample-pair-nn*)
if *auth.Alice* ∈ *s-act1*
| *sil-2′-0*: *S-il* (*map-spmf* (λ*yk*. ((*Some* (**g** [^] *x*, **g** [^] *fst yk*), ()), (*True*, (*EState-Collect*, *s-act1*), *EState-Void*, *s-act2*), *key.State-Store* (*snd yk*), {})) *isample-pair-nk*)
(*map-spmf* (λ*yz*. ((*Some* (**g** [^] *snd yz*, **g** [^] (*x* :: *nat*), **g** [^] *fst yz*), *CState-Half 0*, *CState-Void*), (*auth.State-Collect* **1**, *s-act1*), *auth.State-Void*, *s-act2*)) *isample-pair-nn*)

if *auth.Alice* ∈ *s-act1*
— (Auth1 =a)@(Auth2 =1)
| *sil-0-1*: *S-il* (*return-spmf* ((*None*, ()), (*False*, (*EState-Void*, *s-act1*), *EState-Store*, *s-act2*), *key.PState-Store*, {}))
(*return-spmf* ((*None*, *CState-Void*, *CState-Half 0*), (*auth.State-Void*, *s-act1*), *auth.State-Store* **1**, *s-act2*))
if *auth.Alice* ∈ *s-act2*
— ../(Auth1 =a)@(Auth2 =1) # wl
| *sil-1-1*: *S-il* (*return-spmf* ((*None*, ()), (*False*, (*EState-Store*, *s-act1*), *EState-Store*, *s-act2*), *key.PState-Store*, {}))
(*return-spmf* ((*None*, *CState-Half 0*, *CState-Half 0*), (*auth.State-Store* **1**, *s-act1*), *auth.State-Store* **1**, *s-act2*))
if *auth.Alice* ∈ *s-act1* **and** *auth.Alice* ∈ *s-act2*
| *sil-2-1*: *S-il* (*map-spmf* (λ*k*. ((*None*, ()), (*True*, (*EState-Collect*, *s-act1*), *EState-Store*, *s-act2*), *key.State-Store k*, *s-actk*)) *isample-key*)
(*return-spmf* ((*None*, *CState-Half 0*, *CState-Half 0*), (*auth.State-Collect* **1**, *s-act1*), *auth.State-Store* **1**, *s-act2*))
if *auth.Alice* ∈ *s-act1* **and** *auth.Alice* ∈ *s-act2* **and** *key.Alice* ∉ *s-actk* **and** *auth.Bob* ∈ *s-act1* ⟷ *key.Bob* ∈ *s-actk*
| *sil-3-1*: *S-il* (*return-spmf* ((*None*, ()), (*True*, (*EState-Collect*, *s-act1*), *EState-Store*, *s-act2*), *key.State-Store k*, *s-actk*))
(*map-spmf* (λ*xy*. ((*Some* (**g** [^] (*z* :: *nat*), **g** [^] *fst xy*, **g** [^] *snd xy*), *CState-Half 0*, *CState-Full* (*0*, **1**)), (*auth.State-Collected*, *s-act1*), *auth.State-Store* **1**, *s-act2*)) *isample-pair-nn*)
if *auth.Alice* ∈ *s-act1* **and** *auth.Alice* ∈ *s-act2* **and** *key.Alice* ∉ *s-actk* **and** *auth.Bob* ∈ *s-act1* **and** *key.Bob* ∈ *s-actk* **and** *k* = **g** [^] *z*
— ../(Auth1 =a)@(Auth2 =1) # look
| *sil-1c-1c*: *S-il* (*return-spmf* ((*Some* (**g** [^] *x*, **g** [^] *y*), ()), (*False*, (*EState-Store*, *s-act1*), *EState-Store*, *s-act2*), *key.PState-Store*, {}))
(*map-spmf* (λ*z*. ((*Some* (**g** [^] *z*, **g** [^] (*x* :: *nat*), **g** [^] (*y* :: *nat*)), *CState-Half 0*, *CState-Half 0*), (*auth.State-Store* **1**, *s-act1*), *auth.State-Store* **1**, *s-act2*)) *isample-nat*)
if *auth.Alice* ∈ *s-act1* **and** *auth.Alice* ∈ *s-act2*
| *sil-2c-1c*: *S-il* (*return-spmf* ((*Some* (**g** [^] *x*, **g** [^] *y*), ()), (*True*, (*EState-Collect*, *s-act1*), *EState-Store*, *s-act2*), *key.State-Store k*, *s-actk*))
(*return-spmf* ((*Some* (**g** [^] *z*, **g** [^] (*x* :: *nat*), **g** [^] (*y* :: *nat*)), *CState-Half 0*, *CState-Half 0*), (*auth.State-Collect* **1**, *s-act1*), *auth.State-Store* **1**, *s-act2*))
if *auth.Alice* ∈ *s-act1* **and** *auth.Alice* ∈ *s-act2* **and** *key.Alice* ∉ *s-actk* **and** *auth.Bob* ∈ *s-act1* ⟷ *key.Bob* ∈ *s-actk* **and** *k* = **g** [^] *z* **and** *z* ∈ *set-spmf isample-nat*
| *sil-3c-1c*: *S-il* (*return-spmf* ((*Some* (**g** [^] *x*, **g** [^] *y*), ()), (*True*, (*EState-Collect*, *s-act1*), *EState-Store*, *s-act2*), *key.State-Store k*, *s-actk*))
(*return-spmf* ((*Some* (**g** [^] (*z* :: *nat*), **g** [^] (*x* :: *nat*), **g** [^] (*y* :: *nat*)), *CState-Half 0*, *CState-Full* (*0*, **1**)), (*auth.State-Collected*, *s-act1*), *auth.State-Store* **1**, *s-act2*))
if *auth.Alice* ∈ *s-act1* **and** *auth.Alice* ∈ *s-act2* **and** *key.Alice* ∉ *s-actk* **and** *auth.Bob* ∈ *s-act1* **and** *key.Bob* ∈ *s-actk* **and** *k* = **g** [^] *z*
— (Auth1 =a)@(Auth2 =2)
| *sil-0-2*: *S-il* (*map-spmf* (λ*k*. ((*None*, ()), (*True*, (*EState-Void*, *s-act1*), *ES-*

tate-Collect, *s-act2*), *key.State-Store k*, {})) *isample-key*)
(*return-spmf* ((*None*, *CState-Void*, *CState-Half 0*), (*auth.State-Void*, *s-act1*), *auth.State-Collect* **1**, *s-act2*))
if *auth.Alice* ∈ *s-act2*
— ../(Auth1 =a)@(Auth2 =2) # wl
| *sil-1-2*: *S-il* (*map-spmf* (λ*k*. ((*None*, ()), (*True*, (*EState-Store*, *s-act1*), *EState-Collect*, *s-act2*), *key.State-Store k*, *s-actk*)) *isample-key*)
(*return-spmf* ((*None*, *CState-Half 0*, *CState-Half 0*), (*auth.State-Store* **1**, *s-act1*), *auth.State-Collect* **1**, *s-act2*))
if *auth.Alice* ∈ *s-act1* **and** *auth.Alice* ∈ *s-act2* **and** *auth.Bob* ∈ *s-act2* ⟷ *key.Alice* ∈ *s-actk* **and** *key.Bob* ∉ *s-actk*
| *sil-2-2*: *S-il* (*map-spmf* (λ*k*. ((*None*, ()), (*True*, (*EState-Collect*, *s-act1*), *EState-Collect*, *s-act2*), *key.State-Store k*, *s-actk*)) *isample-key*)
(*return-spmf* ((*None*, *CState-Half 0*, *CState-Half 0*), (*auth.State-Collect* **1**, *s-act1*), *auth.State-Collect* **1**, *s-act2*))
if *auth.Alice* ∈ *s-act1* **and** *auth.Alice* ∈ *s-act2* **and** *auth.Bob* ∈ *s-act2* ⟷ *key.Alice* ∈ *s-actk* **and** *auth.Bob* ∈ *s-act1* ⟷ *key.Bob* ∈ *s-actk*
| *sil-3-2*: *S-il* (*return-spmf* ((*None*, ()), (*True*, (*EState-Collect*, *s-act1*), *EState-Collect*, *s-act2*), *key.State-Store k*, *s-actk*))
(*map-spmf* (λ*xy*. ((*Some* (**g** [^] (*z* :: *nat*), **g** [^] *fst xy*, **g** [^] *snd xy*), *CState-Half 0*, *CState-Full* (*0*, **1**)), (*auth.State-Collected*, *s-act1*), *auth.State-Collect* **1**, *s-act2*)) *isample-pair-nn*)
if *auth.Alice* ∈ *s-act1* **and** *auth.Alice* ∈ *s-act2* **and** *auth.Bob* ∈ *s-act2* ⟷ *key.Alice* ∈ *s-actk* **and** *auth.Bob* ∈ *s-act1* **and** *key.Bob* ∈ *s-actk* **and** *k* = **g** [^] *z*
— ../(Auth1 =a)@(Auth2 =2) # look
| *sil-1c-2c*: *S-il* (*return-spmf* ((*Some* (**g** [^] *x*, **g** [^] *y*), ()), (*True*, (*EState-Store*, *s-act1*), *EState-Collect*, *s-act2*), *key.State-Store k*, *s-actk*))
(*return-spmf* ((*Some* (**g** [^] *z*, **g** [^] (*x* :: *nat*), **g** [^] (*y* :: *nat*)), *CState-Half 0*, *CState-Half 0*), (*auth.State-Store* **1**, *s-act1*), *auth.State-Collect* **1**, *s-act2*))
if *auth.Alice* ∈ *s-act1* **and** *auth.Alice* ∈ *s-act2* **and** *auth.Bob* ∈ *s-act2* ⟷ *key.Alice* ∈ *s-actk* **and** *key.Bob* ∉ *s-actk* **and** *k* = **g** [^] *z* **and** *z* ∈ *set-spmf isample-nat*
| *sil-2c-2c*: *S-il* (*return-spmf* ((*Some* (**g** [^] *x*, **g** [^] *y*), ()), (*True*, (*EState-Collect*, *s-act1*), *EState-Collect*, *s-act2*), *key.State-Store k*, *s-actk*))
(*return-spmf* ((*Some* (**g** [^] *z*, **g** [^] (*x* :: *nat*), **g** [^] (*y* :: *nat*)), *CState-Half 0*, *CState-Half 0*), (*auth.State-Collect* **1**, *s-act1*), *auth.State-Collect* **1**, *s-act2*))
if *auth.Alice* ∈ *s-act1* **and** *auth.Alice* ∈ *s-act2* **and** *auth.Bob* ∈ *s-act2* ⟷ *key.Alice* ∈ *s-actk* **and** *auth.Bob* ∈ *s-act1* ⟷ *key.Bob* ∈ *s-actk* **and** *k* = **g** [^] *z* **and** *z* ∈ *set-spmf isample-nat*
| *sil-3c-2c*: *S-il* (*return-spmf* ((*Some* (**g** [^] *x*, **g** [^] *y*), ()), (*True*, (*EState-Collect*, *s-act1*), *EState-Collect*, *s-act2*), *key.State-Store k*, *s-actk*))
(*return-spmf* ((*Some* (**g** [^] (*z* :: *nat*), **g** [^] (*x* :: *nat*), **g** [^] (*y* :: *nat*)), *CState-Half 0*, *CState-Full* (*0*, **1**)), (*auth.State-Collected*, *s-act1*), *auth.State-Collect* **1**, *s-act2*))
if *auth.Alice* ∈ *s-act1* **and** *auth.Alice* ∈ *s-act2* **and** *auth.Bob* ∈ *s-act2* ⟷ *key.Alice* ∈ *s-actk* **and** *auth.Bob* ∈ *s-act1* **and** *key.Bob* ∈ *s-actk* **and** *k* = **g** [^] *z*
— (Auth1 =a)@(Auth2 =3)
— ../(Auth1 =a)@(Auth2 =3) # wl
| *sil-1-3*: *S-il* (*return-spmf* ((*None*, ()), (*True*, (*EState-Store*, *s-act1*), *EState-Collect*,

s-act2), *key.State-Store k*, *s-actk*))
(*map-spmf* (λ*xy*. ((*Some* (**g** [^] (*z* :: *nat*), **g** [^] *fst xy*, **g** [^] *snd xy*), *CState-Full* (*0*, **1**), *CState-Half 0*), (*auth.State-Store* **1**, *s-act1*), *auth.State-Collected*, *s-act2*)) *isample-pair-nn*)
if *auth.Alice* ∈ *s-act1* **and** *auth.Alice* ∈ *s-act2* **and** *auth.Bob* ∈ *s-act2* **and** *key.Alice* ∈ *s-actk* **and** *key.Bob* ∉ *s-actk* **and** *k* = **g** [^] *z*
| *sil-2-3*: *S-il* (*return-spmf* ((*None*, ()), (*True*, (*EState-Collect*, *s-act1*), *EState-Collect*, *s-act2*), *key.State-Store k*, *s-actk*))
(*map-spmf* (λ*xy*. ((*Some* (**g** [^] (*z* :: *nat*), **g** [^] *fst xy*, **g** [^] *snd xy*), *CState-Full* (*0*, **1**), *CState-Half 0*), (*auth.State-Collect* **1**, *s-act1*), *auth.State-Collected*, *s-act2*)) *isample-pair-nn*)
if *auth.Alice* ∈ *s-act1* **and** *auth.Alice* ∈ *s-act2* **and** *auth.Bob* ∈ *s-act2* **and** *key.Alice* ∈ *s-actk* **and** *auth.Bob* ∈ *s-act1* ⟷ *key.Bob* ∈ *s-actk* **and** *k* = **g** [^] *z*
| *sil-3-3*: *S-il* (*return-spmf* ((*None*, ()), (*True*, (*EState-Collect*, *s-act1*), *EState-Collect*, *s-act2*), *key.State-Store k*, *s-actk*))
(*map-spmf* (λ*xy*. ((*Some* (**g** [^] (*z* :: *nat*), **g** [^] *fst xy*, **g** [^] *snd xy*), *CState-Full* (*0*, **1**), *CState-Full* (*0*, **1**)), (*auth.State-Collected*, *s-act1*), *auth.State-Collected*, *s-act2*)) *isample-pair-nn*)
if *auth.Alice* ∈ *s-act1* **and** *auth.Alice* ∈ *s-act2* **and** *auth.Bob* ∈ *s-act2* **and** *key.Alice* ∈ *s-actk* **and** *auth.Bob* ∈ *s-act1* **and** *key.Bob* ∈ *s-actk* **and** *k* = **g** [^] *z*
— ../(Auth1 =a)@(Auth2 =3) # look
| *sil-1c-3c*: *S-il* (*return-spmf* ((*Some* (**g** [^] *x*, **g** [^] *y*), ()), (*True*, (*EState-Store*, *s-act1*), *EState-Collect*, *s-act2*), *key.State-Store k*, *s-actk*))
(*return-spmf* ((*Some* (**g** [^] (*z* :: *nat*), **g** [^] (*x* :: *nat*), **g** [^] (*y* :: *nat*)), *CState-Full* (*0*, **1**), *CState-Half 0*), (*auth.State-Store* **1**, *s-act1*), *auth.State-Collected*, *s-act2*))
if *auth.Alice* ∈ *s-act1* **and** *auth.Alice* ∈ *s-act2* **and** *auth.Bob* ∈ *s-act2* **and** *key.Alice* ∈ *s-actk* **and** *key.Bob* ∉ *s-actk* **and** *k* = **g** [^] *z*
| *sil-2c-3c*: *S-il* (*return-spmf* ((*Some* (**g** [^] *x*, **g** [^] *y*), ()), (*True*, (*EState-Collect*, *s-act1*), *EState-Collect*, *s-act2*), *key.State-Store k*, *s-actk*))
(*return-spmf* ((*Some* (**g** [^] (*z* :: *nat*), **g** [^] (*x* :: *nat*), **g** [^] (*y* :: *nat*)), *CState-Full* (*0*, **1**), *CState-Half 0*), (*auth.State-Collect* **1**, *s-act1*), *auth.State-Collected*, *s-act2*))
if *auth.Alice* ∈ *s-act1* **and** *auth.Alice* ∈ *s-act2* **and** *auth.Bob* ∈ *s-act2* **and** *key.Alice* ∈ *s-actk* **and** *auth.Bob* ∈ *s-act1* ⟷ *key.Bob* ∈ *s-actk* **and** *k* = **g** [^] *z*
| *sil-3c-3c*: *S-il* (*return-spmf* ((*Some* (**g** [^] *x*, **g** [^] *y*), ()), (*True*, (*EState-Collect*, *s-act1*), *EState-Collect*, *s-act2*), *key.State-Store k*, *s-actk*))
(*return-spmf* ((*Some* (**g** [^] (*z* :: *nat*), **g** [^] (*x* :: *nat*), **g** [^] (*y* :: *nat*)), *CState-Full* (*0*, **1**), *CState-Full* (*0*, **1**)), (*auth.State-Collected*, *s-act1*), *auth.State-Collected*, *s-act2*))
if *auth.Alice* ∈ *s-act1* **and** *auth.Alice* ∈ *s-act2* **and** *auth.Bob* ∈ *s-act2* **and** *key.Alice* ∈ *s-actk* **and** *auth.Bob* ∈ *s-act1* **and** *key.Bob* ∈ *s-actk* **and** *k* = **g** [^] *z*
— (Auth1 =a)@(Auth2 =1')
| *sil-0-1′*: *S-il* (*map-spmf* (λ*x*. ((*Some* (**g** [^] *x*, **g** [^] *y*), ()), (*False*, (*EState-Void*, *s-act1*), *EState-Store*, *s-act2*), *key.PState-Store*, {})) *isample-nat*)
(*map-spmf* (λ*xz*. ((*Some* (**g** [^] *snd xz*, **g** [^] *fst xz*, **g** [^] (*y* :: *nat*)), *CState-Void*, *CState-Half 0*), (*auth.State-Void*, *s-act1*), *auth.State-Store* **1**, *s-act2*)) *isample-pair-nn*)
if *auth.Alice* ∈ *s-act2*
— (Auth1 =a)@(Auth2 =2')
| *sil-0-2′*: *S-il* (*map-spmf* (λ*xk*. ((*Some* (**g** [^] *fst xk*, **g** [^] *y*), ()), (*True*,

(*EState-Void*, *s-act1*), *EState-Collect*, *s-act2*), *key.State-Store* (*snd xk*), {})) *isample-pair-nk*)
(*map-spmf* (λ*xz*. ((*Some* (**g** [^] *snd xz*, **g** [^] *fst xz*, **g** [^] (*y* :: *nat*)), *CState-Void*, *CState-Half 0*), (*auth.State-Void*, *s-act1*), *auth.State-Collect* **1**, *s-act2*)) *isample-pair-nn*)
if *auth.Alice* ∈ *s-act2*

private lemma *trac-eq-core-il*: *trace-core-eq ideal-core′* (*lazy-core DH1-sample*)
((*UNIV* <+> *UNIV*) <+> *UNIV* <+> *UNIV*) ((*UNIV* <+> *UNIV* <+> *UNIV*) <+> *UNIV* <+> *UNIV* <+> *UNIV*) (*UNIV* <+> *UNIV*)
(*return-spmf ideal-s-core′*) (*return-spmf basic-core-sinit*)
⟨*proof*⟩

lemma *connect-ideal*: *connect D* (*obsf-resource ideal-resource*) =
connect D (*obsf-resource* (*RES* (*fused-resource.fuse* (*lazy-core DH1-sample*) *lazy-rest*) (*basic-core-sinit*, *basic-rest-sinit*)))
⟨*proof*⟩

end

12.9 Proving the trace-equivalence of simplified Real and Lazy constructions

context
begin

private abbreviation *rsample-nat* ≡ *sample-uniform* (*order* *G*)
private abbreviation *rsample-pair-nn* ≡ *pair-spmf rsample-nat rsample-nat*

private inductive *S-rl* :: ((*unit* × *′grp cstate* × *′grp cstate*) × *′grp auth.state* × *′grp auth.state*) *spmf*
⇒ ((*′grp st-state* × *′grp cstate* × *′grp cstate*) × *′grp auth.state* × *′grp auth.state*) *spmf* ⇒ *bool*
where
— (Auth1 =a)@(Auth2 =0)
srl-0-0: *S-rl* (*return-spmf* ((), *CState-Void*, *CState-Void*), (*auth.State-Void*, *s-act1*), *auth.State-Void*, *s-act2*))
(*return-spmf* ((*None*, *CState-Void*, *CState-Void*), (*auth.State-Void*, *s-act1*), (*auth.State-Void*, *s-act2*)))
— ../(Auth1 =a)@(Auth2 =0) # wl
| *srl-1-0*: *S-rl* (*map-spmf* (λ*x*. ((), *CState-Half x*, *CState-Void*), (*auth.State-Store* (**g** [^] *x*), *s-act1*), *auth.State-Void*, *s-act2*)) *rsample-nat*)
(*return-spmf* ((*None*, *CState-Half 0*, *CState-Void*), (*auth.State-Store* **1**, *s-act1*), *auth.State-Void*, *s-act2*))
| *srl-2-0*: *S-rl* (*map-spmf* (λ*x*. ((), *CState-Half x*, *CState-Void*), (*auth.State-Collect* (**g** [^] *x*), *s-act1*), *auth.State-Void*, *s-act2*)) *rsample-nat*)
(*return-spmf* ((*None*, *CState-Half 0*, *CState-Void*), (*auth.State-Collect* **1**, *s-act1*), *auth.State-Void*, *s-act2*))
— ../(Auth1 =a)@(Auth2 =0) # look

| *srl-1′-0*: *S-rl* (*return-spmf* ((), *CState-Half x*, *CState-Void*), (*auth.State-Store* (**g** [^] *x*), *s-act1*), *auth.State-Void*, *s-act2*))
(*map-spmf* (λ*y*. ((*Some* ((**g** [^] *x*) [^] *y*, **g** [^] *x*, **g** [^] *y*), *CState-Half 0*, *CState-Void*), (*auth.State-Store* **1**, *s-act1*), *auth.State-Void*, *s-act2*)) *rsample-nat*)
| *srl-2′-0*: *S-rl* (*return-spmf* ((), *CState-Half x*, *CState-Void*), (*auth.State-Collect* (**g** [^] *x*), *s-act1*), *auth.State-Void*, *s-act2*))
(*map-spmf* (λ*y*. ((*Some* ((**g** [^] *x*) [^] *y*, **g** [^] *x*, **g** [^] *y*), *CState-Half 0*, *CState-Void*), (*auth.State-Collect* **1**, *s-act1*), *auth.State-Void*, *s-act2*)) *rsample-nat*)
— (Auth1 =a)@(Auth2 =1)
| *srl-0-1*: *S-rl* (*map-spmf* (λ*y*. ((), *CState-Void*, *CState-Half y*), (*auth.State-Void*, *s-act1*), *auth.State-Store* (**g** [^] *y*), *s-act2*)) *rsample-nat*)
(*return-spmf* ((*None*, *CState-Void*, *CState-Half 0*), (*auth.State-Void*, *s-act1*), *auth.State-Store* **1**, *s-act2*))
— ../(Auth1 =a)@(Auth2 =1) # wl
| *srl-1-1*: *S-rl* (*map-spmf* (λ*yx*. ((), *CState-Half* (*snd yx*), *CState-Half* (*fst yx*)), (*auth.State-Store* (**g** [^] *snd yx*), *s-act1*), *auth.State-Store* (**g** [^] *fst yx*), *s-act2*)) *rsample-pair-nn*)
(*return-spmf* ((*None*, *CState-Half 0*, *CState-Half 0*), (*auth.State-Store* **1**, *s-act1*), *auth.State-Store* **1**, *s-act2*))
| *srl-2-1*: *S-rl* (*map-spmf* (λ*yx*. ((), *CState-Half* (*snd yx*), *CState-Half* (*fst yx*)), (*auth.State-Collect* (**g** [^] *snd yx*), *s-act1*), *auth.State-Store* (**g** [^] *fst yx*), *s-act2*)) *rsample-pair-nn*)
(*return-spmf* ((*None*, *CState-Half 0*, *CState-Half 0*), (*auth.State-Collect* **1**, *s-act1*), *auth.State-Store* **1**, *s-act2*))
— ../(Auth1 =a)@(Auth2 =1) # look
| *srl-1c-1c*: *S-rl* (*return-spmf* ((), *CState-Half x*, *CState-Half y*), (*auth.State-Store* (**g** [^] *x*), *s-act1*), *auth.State-Store* (**g** [^] *y*), *s-act2*))
(*return-spmf* ((*Some* ((**g** [^] *x*) [^] *y*, **g** [^] *x*, **g** [^] *y*), *CState-Half 0*, *CState-Half 0*), (*auth.State-Store* **1**, *s-act1*), *auth.State-Store* **1**, *s-act2*))
| *srl-2c-1c*: *S-rl* (*return-spmf* ((), *CState-Half x*, *CState-Half y*), (*auth.State-Collect* (**g** [^] *x*), *s-act1*), *auth.State-Store* (**g** [^] *y*), *s-act2*))
(*return-spmf* ((*Some* ((**g** [^] *x*) [^] *y*, **g** [^] *x*, **g** [^] *y*), *CState-Half 0*, *CState-Half 0*), (*auth.State-Collect* **1**, *s-act1*), *auth.State-Store* **1**, *s-act2*))
| *srl-3c-1c*: *S-rl* (*return-spmf* ((), *CState-Half x*, *CState-Full* (*y*, *z*)), (*auth.State-Collected*, *s-act1*), *auth.State-Store* (**g** [^] *y*), *s-act2*))
(*return-spmf* ((*Some* (*z*, **g** [^] *x*, **g** [^] *y*), *CState-Half 0*, *CState-Full* (*0*, **1**)), (*auth.State-Collected*, *s-act1*), *auth.State-Store* **1**, *s-act2*))
if *z* = (**g** [^] *x*) [^] *y*
— (Auth1 =a)@(Auth2 =2)
| *srl-0-2*: *S-rl* (*map-spmf* (λ*y*. ((), *CState-Void*, *CState-Half y*), (*auth.State-Void*, *s-act1*), *auth.State-Collect* (**g** [^] *y*), *s-act2*)) *rsample-nat*)
(*return-spmf* ((*None*, *CState-Void*, *CState-Half 0*), (*auth.State-Void*, *s-act1*), *auth.State-Collect* **1**, *s-act2*))
— ../(Auth1 =a)@(Auth2 =2) # wl
| *srl-1-2*: *S-rl* (*map-spmf* (λ*yx*. ((), *CState-Half* (*snd yx*), *CState-Half* (*fst yx*)), (*auth.State-Store* (**g** [^] *snd yx*), *s-act1*), *auth.State-Collect* (**g** [^] *fst yx*), *s-act2*)) *rsample-pair-nn*)
(*return-spmf* ((*None*, *CState-Half 0*, *CState-Half 0*), (*auth.State-Store* **1**, *s-act1*), *auth.State-Collect* **1**, *s-act2*))

| *srl-2-2*: *S-rl* (*map-spmf* (λ*yx*. (((), *CState-Half* (*snd yx*), *CState-Half* (*fst yx*)), (*auth.State-Collect* (**g** [^] *snd yx*), *s-act1*), *auth.State-Collect* (**g** [^] *fst yx*), *s-act2*)) *rsample-pair-nn*)
(*return-spmf* ((*None*, *CState-Half 0*, *CState-Half 0*), (*auth.State-Collect* **1**, *s-act1*), *auth.State-Collect* **1**, *s-act2*))
— ../(Auth1 =a)@(Auth2 =2) # look
| *srl-1c-2c*: *S-rl* (*return-spmf* (((), *CState-Half x*, *CState-Half y*), (*auth.State-Store* (**g** [^] *x*), *s-act1*), *auth.State-Collect* (**g** [^] *y*), *s-act2*))
(*return-spmf* ((*Some* ((**g** [^] *x*) [^] *y*, **g** [^] *x*, **g** [^] *y*), *CState-Half 0*, *CState-Half 0*), (*auth.State-Store* **1**, *s-act1*), *auth.State-Collect* **1**, *s-act2*))
| *srl-2c-2c*: *S-rl* (*return-spmf* (((), *CState-Half x*, *CState-Half y*), (*auth.State-Collect* (**g** [^] *x*), *s-act1*), *auth.State-Collect* (**g** [^] *y*), *s-act2*))
(*return-spmf* ((*Some* ((**g** [^] *x*) [^] *y*, **g** [^] *x*, **g** [^] *y*), *CState-Half 0*, *CState-Half 0*), (*auth.State-Collect* **1**, *s-act1*), *auth.State-Collect* **1**, *s-act2*))
| *srl-3c-2c*: *S-rl* (*return-spmf* (((), *CState-Half x*, *CState-Full* (*y*, *z*)), (*auth.State-Collected*, *s-act1*), *auth.State-Collect* (**g** [^] *y*), *s-act2*))
(*return-spmf* ((*Some* (*z*, **g** [^] *x*, **g** [^] *y*), *CState-Half 0*, *CState-Full* (*0*, **1**)), (*auth.State-Collected*, *s-act1*), *auth.State-Collect* **1**, *s-act2*))
if *z* = (**g** [^] *x*) [^] *y*
— (Auth1 =a)@(Auth2 =3)
| *srl-1c-3c*: *S-rl* (*return-spmf* (((), *CState-Full* (*x*, *z*), *CState-Half y*), (*auth.State-Store* (**g** [^] *x*), *s-act1*), *auth.State-Collected*, *s-act2*))
(*return-spmf* ((*Some* (*z*, **g** [^] *x*, **g** [^] *y*), *CState-Full* (*0*, **1**), *CState-Half 0*), (*auth.State-Store* **1**, *s-act1*), *auth.State-Collected*, *s-act2*))
if *z* = (**g** [^] *y*) [^] *x*
| *srl-2c-3c*: *S-rl* (*return-spmf* (((), *CState-Full* (*x*, *z*), *CState-Half y*), (*auth.State-Collect* (**g** [^] *x*), *s-act1*), *auth.State-Collected*, *s-act2*))
(*return-spmf* ((*Some* (*z*, **g** [^] *x*, **g** [^] *y*), *CState-Full* (*0*, **1**), *CState-Half 0*), (*auth.State-Collect* **1**, *s-act1*), *auth.State-Collected*, *s-act2*))
if *z* = (**g** [^] *y*) [^] *x*
| *srl-3c-3c*: *S-rl* (*return-spmf* (((), *CState-Full* (*x*, *z*), *CState-Full* (*y*, *z*)), (*auth.State-Collected*, *s-act1*), *auth.State-Collected*, *s-act2*))
(*return-spmf* ((*Some* (*z*, **g** [^] *x*, **g** [^] *y*), *CState-Full* (*0*, **1**), *CState-Full* (*0*, **1**)), (*auth.State-Collected*, *s-act1*), *auth.State-Collected*, *s-act2*))
if *z* = (**g** [^] *y*) [^] *x*
— (Auth1 =0)@(Auth2 =1')
| *srl-0-1'*: *S-rl* (*return-spmf* (((), *CState-Void*, *CState-Half y*), (*auth.State-Void*, *s-act1*), *auth.State-Store* (**g** [^] *y*), *s-act2*))
(*map-spmf* (λ*x*. ((*Some* ((**g** [^] *x*) [^] *y*, **g** [^] *x*, **g** [^] *y*), *CState-Void*, *CState-Half 0*), (*auth.State-Void*, *s-act1*), *auth.State-Store* **1**, *s-act2*)) *rsample-nat*)
— (Auth1 =0)@(Auth2 =2')
| *srl-0-2'*: *S-rl* (*return-spmf* (((), *CState-Void*, *CState-Half y*), (*auth.State-Void*, *s-act1*), *auth.State-Collect* (**g** [^] *y*), *s-act2*))
(*map-spmf* (λ*x*. ((*Some* ((**g** [^] *x*) [^] *y*, **g** [^] *x*, **g** [^] *y*), *CState-Void*, *CState-Half 0*), (*auth.State-Void*, *s-act1*), *auth.State-Collect* **1**, *s-act2*)) *rsample-nat*)

private lemma *trac-eq-core-rl*: *trace-core-eq real-core'* (*basic-core DH0-sample*)
(*UNIV* <+> *UNIV*) ((*UNIV* <+> *UNIV* <+> *UNIV*) <+> *UNIV* <+>

UNIV <+> *UNIV*) ((*UNIV* <+> *UNIV*) <+> *UNIV* <+> *UNIV*)
(*return-spmf real-s-core'*) (*return-spmf basic-core-sinit*)
⟨*proof*⟩

lemma *trace-eq-fuse-rl*: *UNIV* $\vdash_R$ 1_C $\models$ $rassocl_C$ ▷ *RES* (*fused-resource.fuse real-core' real-rest'*) (*real-s-core'*, *real-s-rest'*)
≈ *RES* (*fused-resource.fuse* (*lazy-core DH0-sample*) *lazy-rest*) (*basic-core-sinit*, *basic-rest-sinit*)
⟨*proof*⟩

lemma *connect-real*: *connect D* (*obsf-resource real-resource*) = *connect D* (*obsf-resource* (*RES* (*fused-resource.fuse* (*lazy-core DH0-sample*) *lazy-rest*) (*basic-core-sinit*, *basic-rest-sinit*)))
⟨*proof*⟩

end

end

end

12.10 Concrete security

context *diffie-hellman* **begin**

context
fixes
auth1-rest :: ('*auth1-s-rest*, *auth.event*, '*auth1-iadv-rest*, '*auth1-iusr-rest*, '*auth1-oadv-rest*, '*auth1-ousr-rest*) *rest-wstate* **and**
auth2-rest :: ('*auth2-s-rest*, *auth.event*, '*auth2-iadv-rest*, '*auth2-iusr-rest*, '*auth2-oadv-rest*, '*auth2-ousr-rest*) *rest-wstate* **and**
$\mathcal{I}$-*adv-rest1* **and** $\mathcal{I}$-*adv-rest2* **and** $\mathcal{I}$-*usr-rest1* **and** $\mathcal{I}$-*usr-rest2* **and** *I-auth1-rest* **and** *I-auth2-rest*
assumes
WT-auth1-rest [*WT-intro*]: *WT-rest* $\mathcal{I}$-*adv-rest1* $\mathcal{I}$-*usr-rest1 I-auth1-rest auth1-rest* **and**
WT-auth2-rest [*WT-intro*]: *WT-rest* $\mathcal{I}$-*adv-rest2* $\mathcal{I}$-*usr-rest2 I-auth2-rest auth2-rest*
begin

theorem *secure*:
defines $\mathcal{I}$-*real* ≡ (($\mathcal{I}$-*full* $\oplus_{\mathcal{I}}$ ($\mathcal{I}$-*full* $\oplus_{\mathcal{I}}$ $\mathcal{I}$-*uniform* (*auth.Inp-Fedit* ‘ (*carrier* $\mathcal{G}$)) *UNIV*)) $\oplus_{\mathcal{I}}$ ($\mathcal{I}$-*full* $\oplus_{\mathcal{I}}$ ($\mathcal{I}$-*full* $\oplus_{\mathcal{I}}$ $\mathcal{I}$-*uniform* (*auth.Inp-Fedit* ‘ (*carrier* $\mathcal{G}$)) *UNIV*))) $\oplus_{\mathcal{I}}$ ($\mathcal{I}$-*adv-rest1* $\oplus_{\mathcal{I}}$ $\mathcal{I}$-*adv-rest2*)
and $\mathcal{I}$-*common* ≡ ($\mathcal{I}$-*uniform UNIV* (*key.Out-Alice* ‘ *carrier* $\mathcal{G}$) $\oplus_{\mathcal{I}}$ $\mathcal{I}$-*uniform UNIV* (*key.Out-Bob* ‘ *carrier* $\mathcal{G}$)) $\oplus_{\mathcal{I}}$ (($\mathcal{I}$-*full* $\oplus_{\mathcal{I}}$ $\mathcal{I}$-*full*) $\oplus_{\mathcal{I}}$ ($\mathcal{I}$-*usr-rest1* $\oplus_{\mathcal{I}}$ $\mathcal{I}$-*usr-rest2*))
and $\mathcal{I}$-*ideal* ≡ $\mathcal{I}$-*full* $\oplus_{\mathcal{I}}$ ($\mathcal{I}$-*full* $\oplus_{\mathcal{I}}$ ($\mathcal{I}$-*adv-rest1* $\oplus_{\mathcal{I}}$ $\mathcal{I}$-*adv-rest2*))
shows *constructive-security-obsf*
(*real-resource TYPE*(-) *TYPE*(-) *auth1-rest auth2-rest*)

(*key.resource* (*ideal-rest auth1-rest auth2-rest*))
(*let sim* = *CNV sim-callee None in* ((*sim* $|_=$ 1_C) $\odot$ $lassocr_C$))
$\mathcal{I}$-*real* $\mathcal{I}$-*ideal* $\mathcal{I}$-*common* $\mathcal{A}$
(*ddh.advantage* $\mathcal{G}$ (*DH-adversary TYPE*(-) *TYPE*(-) *auth1-rest auth2-rest* $\mathcal{A}$))
⟨*proof*⟩

end

end

12.11 Asymptotic security

locale *diffie-hellman′* =
fixes $\mathcal{G}$ :: *security* ⇒ *'grp cyclic-group*
assumes *diffie-hellman* [*locale-witness*]: $\bigwedge\eta$. *diffie-hellman* ($\mathcal{G}$ η)
begin

sublocale *diffie-hellman* $\mathcal{G}$ η **for** η ⟨*proof*⟩

definition *real-resource′* **where** *real-resource′ rest1 rest2* η = *real-resource TYPE*(-) *TYPE*(-) η (*rest1* η) (*rest2* η)
definition *ideal-resource′* **where** *ideal-resource′ rest1 rest2* η = *key.resource* η (*ideal-rest* (*rest1* η) (*rest2* η))
definition *sim′* **where** *sim′* η = (*let sim* = *CNV* (*sim-callee* η) *None in* ((*sim* $|_=$ 1_C) $\odot$ $lassocr_C$))

context
fixes
auth1-rest :: *nat* ⇒ (*'auth1-s-rest*, *auth.event*, *'auth1-iadv-rest*, *'auth1-iusr-rest*, *'auth1-oadv-rest*, *'auth1-ousr-rest*) *rest-wstate* **and**
auth2-rest :: *nat* ⇒ (*'auth2-s-rest*, *auth.event*, *'auth2-iadv-rest*, *'auth2-iusr-rest*, *'auth2-oadv-rest*, *'auth2-ousr-rest*) *rest-wstate* **and**
$\mathcal{I}$-*adv-rest1* **and** $\mathcal{I}$-*adv-rest2* **and** $\mathcal{I}$-*usr-rest1* **and** $\mathcal{I}$-*usr-rest2* **and** *I-auth1-rest* **and** *I-auth2-rest*
assumes
WT-auth1-rest: $\bigwedge\eta$. *WT-rest* ($\mathcal{I}$-*adv-rest1* η) ($\mathcal{I}$-*usr-rest1* η) (*I-auth1-rest* η) (*auth1-rest* η) **and**
WT-auth2-rest: $\bigwedge\eta$. *WT-rest* ($\mathcal{I}$-*adv-rest2* η) ($\mathcal{I}$-*usr-rest2* η) (*I-auth2-rest* η) (*auth2-rest* η)
begin

theorem *secure*:
defines $\mathcal{I}$-*real* ≡ $\lambda\eta$. (($\mathcal{I}$-*full* $\oplus_\mathcal{I}$ ($\mathcal{I}$-*full* $\oplus_\mathcal{I}$ $\mathcal{I}$-*uniform* (*auth.Inp-Fedit* ‘ (*carrier* ($\mathcal{G}$ η))) *UNIV*)) $\oplus_\mathcal{I}$ ($\mathcal{I}$-*full* $\oplus_\mathcal{I}$ ($\mathcal{I}$-*full* $\oplus_\mathcal{I}$ $\mathcal{I}$-*uniform* (*auth.Inp-Fedit* ‘ (*carrier* ($\mathcal{G}$ η))) *UNIV*))) $\oplus_\mathcal{I}$ ($\mathcal{I}$-*adv-rest1* η $\oplus_\mathcal{I}$ $\mathcal{I}$-*adv-rest2* η)
and $\mathcal{I}$-*common* ≡ $\lambda\eta$. ($\mathcal{I}$-*uniform UNIV* (*key.Out-Alice* ‘ *carrier* ($\mathcal{G}$ η)) $\oplus_\mathcal{I}$ $\mathcal{I}$-*uniform UNIV* (*key.Out-Bob* ‘ *carrier* ($\mathcal{G}$ η))) $\oplus_\mathcal{I}$ (($\mathcal{I}$-*full* $\oplus_\mathcal{I}$ $\mathcal{I}$-*full*) $\oplus_\mathcal{I}$ ($\mathcal{I}$-*usr-rest1* η $\oplus_\mathcal{I}$ $\mathcal{I}$-*usr-rest2* η))
and $\mathcal{I}$-*ideal* ≡ $\lambda\eta$. $\mathcal{I}$-*full* $\oplus_\mathcal{I}$ ($\mathcal{I}$-*full* $\oplus_\mathcal{I}$ ($\mathcal{I}$-*adv-rest1* η $\oplus_\mathcal{I}$ $\mathcal{I}$-*adv-rest2* η))

assumes *DDH*: *negligible* (λη. *ddh.advantage* (*G* η) (*DH-adversary TYPE*(-) *TYPE*(-) η (*auth1-rest* η) (*auth2-rest* η) (*A* η)))
shows *constructive-security-obsf'* (*real-resource' auth1-rest auth2-rest*) (*ideal-resource' auth1-rest auth2-rest*) *sim' I-real I-ideal I-common A*
⟨*proof*⟩

end

end

end
theory *DH-OTP* **imports**
One-Time-Pad
Diffie-Hellman-CC
begin

We need both a group structure and a boolean algebra. Unfortunately, records allow only one extension slot, so we can't have just a single structure with both operations.

context *diffie-hellman* **begin**

lemma *WT-ideal-rest* [*WT-intro*]:
assumes *WT-auth1-rest* [*WT-intro*]: *WT-rest I-adv-rest1 I-usr-rest1 I-auth1-rest auth1-rest*
and *WT-auth2-rest* [*WT-intro*]: *WT-rest I-adv-rest2 I-usr-rest2 I-auth2-rest auth2-rest*
shows *WT-rest* (*I-full* ⊕$_I$ (*I-adv-rest1* ⊕$_I$ *I-adv-rest2*)) ((*I-full* ⊕$_I$ *I-full*) ⊕$_I$ (*I-usr-rest1* ⊕$_I$ *I-usr-rest2*))
(λ(-, *s*). *pred-prod I-auth1-rest I-auth2-rest s*) (*ideal-rest auth1-rest auth2-rest*)
⟨*proof*⟩

end

locale *dh-otp* = *dh*: *diffie-hellman G* + *otp*: *one-time-pad L*
for *G* :: *'grp cyclic-group*
and *L* :: *'grp boolean-algebra* +
assumes *carrier-G-L*: *carrier G* = *carrier L*
begin

theorem *secure*:
assumes *WT-rest I-adv-resta I-usr-resta I-auth-rest auth-rest*
and *WT-rest I-adv-rest1 I-usr-rest1 I-auth1-rest auth1-rest*
and *WT-rest I-adv-rest2 I-usr-rest2 I-auth2-rest auth2-rest*
shows
constructive-security-obsf
(1$_C$ |= *wiring-c1r22-c1r22* (*CNV otp.enc-callee* ()) (*CNV otp.dec-callee* ()) |= 1$_C$ ▷
fused-wiring ▷ *diffie-hellman.real-resource G auth1-rest auth2-rest* ‖ *dh.auth.resource*

auth-rest)
(*otp.sec.resource* (*otp.ideal-rest* (*dh.ideal-rest auth1-rest auth2-rest*) *auth-rest*))
((*1*$_C$ ⊙
(*parallel-wiring* ⊙ ((*let sim* = *CNV dh.sim-callee None in* (*sim* |= *1*$_C$) ⊙ *lassocr*$_C$) |= *1*$_C$) ⊙ *parallel-wiring*) ⊙
1$_C$) ⊙
(*otp.sim* |= *1*$_C$))
((((*I-full* ⊕$_\mathcal{I}$ (*I-full* ⊕$_\mathcal{I}$ *I-uniform* (*otp.sec.Inp-Fedit* ‘ *carrier* $\mathcal{G}$) *UNIV*)) ⊕$_\mathcal{I}$
(*I-full* ⊕$_\mathcal{I}$ (*I-full* ⊕$_\mathcal{I}$ *I-uniform* (*otp.sec.Inp-Fedit* ‘ *carrier* $\mathcal{G}$) *UNIV*)))
⊕$_\mathcal{I}$
(*I-full* ⊕$_\mathcal{I}$ (*I-full* ⊕$_\mathcal{I}$ *I-uniform* (*otp.sec.Inp-Fedit* ‘ *carrier* $\mathcal{L}$) *UNIV*))) ⊕$_\mathcal{I}$
((*I-adv-rest1* ⊕$_\mathcal{I}$ *I-adv-rest2*) ⊕$_\mathcal{I}$ *I-adv-resta*))
((*I-full* ⊕$_\mathcal{I}$ (*I-full* ⊕$_\mathcal{I}$ *I-uniform* (*otp.sec.Inp-Fedit* ‘ *carrier* $\mathcal{L}$) *UNIV*)) ⊕$_\mathcal{I}$
((*I-full* ⊕$_\mathcal{I}$ (*I-adv-rest1* ⊕$_\mathcal{I}$ *I-adv-rest2*)) ⊕$_\mathcal{I}$ *I-adv-resta*))
((*I-uniform* (*otp.sec.Inp-Send* ‘ *carrier* $\mathcal{L}$) *UNIV* ⊕$_\mathcal{I}$ *I-uniform UNIV*
(*otp.sec.Out-Recv* ‘ *carrier* $\mathcal{L}$)) ⊕$_\mathcal{I}$
(((*I-full* ⊕$_\mathcal{I}$ *I-full*) ⊕$_\mathcal{I}$ (*I-usr-rest1* ⊕$_\mathcal{I}$ *I-usr-rest2*)) ⊕$_\mathcal{I}$ *I-usr-resta*))
$\mathcal{A}$ (*0* + (*ddh.advantage* $\mathcal{G}$
(*diffie-hellman.DH-adversary* $\mathcal{G}$ *auth1-rest auth2-rest*
(*absorb*
(*absorb* $\mathcal{A}$
(*obsf-converter* (*1*$_C$ |= *wiring-c1r22-c1r22* (*CNV otp.enc-callee* ()) (*CNV otp.dec-callee* ()) |= *1*$_C$)))
(*obsf-converter*
(*fused-wiring* ⊙ (*1*$_C$ |$_\propto$ *converter-of-resource* (*1*$_C$ |= *1*$_C$ ▷ *dh.auth.resource auth-rest*))))))) +
0))
⟨*proof*⟩

end

end

References

[1] D. A. Basin, A. Lochbihler, U. Maurer, and S. R. Sefidgar. Abstract modeling of systems communication in constructive cryptography using CryptHOL. 2021. http://www.andreas-lochbihler.de/pub/basin2021.pdf, Draft paper.

[2] D. A. Basin, A. Lochbihler, and S. R. Sefidgar. CryptHOL: Game-based proofs in higher-order logic. *Journal of Cryptology*, 33(2):494–566, 2020.

[3] A. Lochbihler. Probabilistic functions and cryptographic oracles in higher order logic. In *European Symposium on Programming (ESOP 2016), Proceedings*, volume 9632 of *LNCS*, pages 503–531. Springer, 2016.

[4] A. Lochbihler. CryptHOL. *Archive of Formal Proofs*, 2017. https://isa-afp.org/entries/CryptHOL.html, Formal proof development.

[5] A. Lochbihler and S. R. Sefidgar. Constructive cryptography in HOL. *Archive of Formal Proofs*, 2018. https://isa-afp.org/entries/Constructive_Cryptography.html, Formal proof development.

[6] A. Lochbihler, S. R. Sefidgar, D. Basin, and U. Maurer. Formalizing constructive cryptography using crypthol. In *Computer Security Foundations Symposium (CSF 2019), Proceedings*, pages 152–166. IEEE, 2019.

[7] U. Maurer. Constructive cryptography - A new paradigm for security definitions and proofs. In *Theory of Security and Applications - Joint Workshop (TOSCA 2011), Revised Selected Papers*, volume 6993 of *LNCS*, pages 33–56. Springer, 2011.

[8] U. Maurer and R. Renner. Abstract cryptography. In *Innovations in Computer Science (ICS 2010), Proceedings*, pages 1–21. Tsinghua University Press, 2011.

[9] U. Maurer and R. Renner. From indifferentiability to constructive cryptography (and back). In *Theory of Cryptography Conference (TCC 2016), Proceedings, Part I*, volume 9985 of *LNCS*, pages 3–24. Springer, 2016.