

# Constructive Cryptography in HOL: the Communication Modeling Aspect

Andreas Lochbihler and S. Reza Sefidgar

February 6, 2026

## 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

<b>1</b>	<b>Material for Isabelle library</b>	<b>4</b>
1.1	Probabilities . . . . .	5
1.1.1	Conditional probabilities . . . . .	7
<b>2</b>	<b>Material for CryptHOL</b>	<b>10</b>
2.1	<i>try-gpv</i> . . . . .	11
2.2	term <i>gpv-stop</i> . . . . .	13
2.3	term <i>exception-<math>\mathcal{I}</math></i> . . . . .	15
2.4	inline . . . . .	16
<b>3</b>	<b>Material for Constructive Crypto</b>	<b>18</b>
3.1	<i>Constructive-Cryptography.Wiring</i> . . . . .	21
3.2	Probabilistic finite converter . . . . .	23
3.3	colossless converter . . . . .	26
3.4	trace equivalence . . . . .	27

<b>4</b>	<b>Fused Resource</b>	<b>38</b>
4.1	Event Oracles – they generate events . . . . .	38
4.2	Event Handlers – they absorb (and silently handle) events . .	39
4.3	Fused Resource Construction . . . . .	40
4.4	More helpful construction functions . . . . .	48
<b>5</b>	<b>Traces</b>	<b>50</b>
<b>6</b>	<b>State Isomorphism</b>	<b>65</b>
6.1	Parallel State Isomorphism . . . . .	66
6.2	Trisplit State Isomorphism . . . . .	66
6.3	Assocl-Swap State Isomorphism . . . . .	66
<b>7</b>	<b>Concrete security definition</b>	<b>72</b>
7.1	Composition theorems . . . . .	73
<b>8</b>	<b>Asymptotic security definition</b>	<b>76</b>
8.1	Composition theorems . . . . .	77
<b>9</b>	<b>Key specification</b>	<b>78</b>
9.1	Data-types for Parties, State, Events, Input, and Output . . .	79
9.1.1	Basic lemmas for automated handling of party sets (i.e. <i>s-shell</i> ) . . . . .	79
9.2	Defining the event handler . . . . .	79
9.3	Defining the adversary interface . . . . .	80
9.4	Defining the user interfaces . . . . .	80
9.5	Defining the Fuse Resource . . . . .	81
9.5.1	Lemma showing that the resulting resource is well-typed	81
<b>10</b>	<b>Channel specification</b>	<b>81</b>
10.1	Data-types for Parties, State, Events, Input, and Output . . .	82
10.1.1	Basic lemmas for automated handling of party sets (i.e. <i>s-shell</i> ) . . . . .	82
10.2	Defining the event handler . . . . .	83
10.3	Defining the adversary interfaces . . . . .	83
10.4	Defining the user interfaces . . . . .	84
10.5	Defining the Fused Resource . . . . .	84
10.5.1	Lemma showing that the resulting resource is well-typed	85
<b>11</b>	<b>One-time-pad construction</b>	<b>85</b>
11.1	Defining user callees . . . . .	86
11.2	Defining adversary converter . . . . .	86
11.3	Defining event-translator . . . . .	87
11.3.1	Basic lemmas for automated handling of <i>sec-party-of-key-party</i>	88
11.4	Defining Ideal and Real constructions . . . . .	88

11.5	Wiring and simplifying the Ideal construction . . . . .	89
11.5.1	The ideal attachment lemma . . . . .	89
11.6	Wiring and simplifying the Real construction . . . . .	89
11.6.1	The real attachment lemma . . . . .	90
11.7	Proving the trace-equivalence of simplified Ideal and Real constructions . . . . .	90
11.7.1	Proving the trace-equivalence of cores . . . . .	91
11.7.2	Proving the trace equivalence of fused cores and rests . . . . .	92
11.7.3	Simplifying the final resource by moving the interfaces from core to rest . . . . .	93
11.8	Concrete security . . . . .	93
11.9	Asymptotic security . . . . .	95
<b>12</b>	<b>Diffie-Hellman construction</b>	<b>96</b>
12.1	Defining user callees . . . . .	96
12.2	Defining adversary callee . . . . .	97
12.3	Defining event-translator . . . . .	98
12.4	Defining Ideal and Real constructions . . . . .	100
12.5	Wiring and simplifying the Ideal construction . . . . .	101
12.5.1	The ideal attachment lemma . . . . .	102
12.6	Wiring and simplifying the Real construction . . . . .	102
12.6.1	The real attachment lemma . . . . .	103
12.7	A lazy construction and its DH reduction . . . . .	103
12.7.1	Defining a lazy construction with an inlined sampler . . . . .	103
12.7.2	Defining a lazy construction with an external sampler . . . . .	105
12.7.3	Reduction to Diffie-Hellman game . . . . .	107
12.8	Proving the trace-equivalence of simplified Ideal and Lazy constructions . . . . .	109
12.9	Proving the trace-equivalence of simplified Real and Lazy constructions . . . . .	113
12.10	Concrete security . . . . .	116
12.11	Asymptotic security . . . . .	117

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

## 1 Material for Isabelle library

**lemma** *eq-alt-conversep*:  $(=) = (BNF-Def.Grp UNIV id)^{-1-1}$   
 $\langle proof \rangle$

**parametric-constant**

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

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

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

**lemma** *insert-Inl-Plus [simp]*:  $insert\ (Inl\ x)\ (A\ <+>\ B) = insert\ x\ A\ <+>\ B$   
 $\langle proof \rangle$

**lemma** *insert-Inr-Plus [simp]*:  $insert\ (Inr\ x)\ (A\ <+>\ B) = A\ <+>\ insert\ x\ B$   
 $\langle proof \rangle$

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

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

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

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

**lemma** *surj-map-sum*:  $surj\ (map\text{-}sum\ f\ g)$  **if**  $surj\ f\ surj\ g$   
 $\langle proof \rangle$

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

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

(*inv-into UNIV g*)  
⟨*proof*⟩

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

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

**lemma** *admissible-le1-mcont [cont-intro]*:  
  *ccpo.admissible lub ord (λx. 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**  $(\int^+ x. indicator \{y\} x * f x \partial M) = emeasure M \{y\} * f y$   
  ⟨*proof*⟩

**lemma** *nn-integral-indicator-singleton1' [simp]*:  
  **assumes**  $\{y\} \in sets M$   
  **shows**  $(\int^+ x. indicator \{x\} y * f x \partial M) = emeasure M \{y\} * f y$   
  ⟨*proof*⟩

## 1.1 Probabilities

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

**lemma** *measure-spmf-cong*:  $measure (measure-spmf p) A = measure (measure-spmf p) B$   
  **if**  $A \cap set-spmf p = B \cap set-spmf p$   
  ⟨*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*:  
   $bind-spmf (map-spmf (to-nat-on (set-spmf p)) p) (\lambda n. f (from-nat-into (set-spmf p) n)) = bind-spmf p f$   
  ⟨*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*:

$\llbracket \text{rel-spmf } R \text{ } p \text{ } q; \text{rel-spmf } S \text{ } q \text{ } r \rrbracket \implies \text{rel-spmf } (R \text{ } OO \text{ } eq\text{-onp } (\lambda x. x \in \text{set-spmf } q) \text{ } OO \text{ } S) \text{ } p \text{ } r$   
⟨proof⟩

**lemma** *mcont2mcont-spmf [cont-intro]*:

*mcont lub ord Sup (≤) (λp. 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  $\implies$  ord-spmf (=) (p  $\upharpoonright$  A) (q  $\upharpoonright$  A)*  
⟨proof⟩

**lemma** *restrict-lub-spmf*:

**assumes** *chain: Complete-Partial-Order.chain (ord-spmf (=)) Y*  
**shows** *restrict-spmf (lub-spmf Y) A = lub-spmf ((λp. 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 (=)) (λp. p  $\upharpoonright$  A)*  
⟨proof⟩

**lemma** *mcont2mcont-restrict-spmf [THEN spmf.mcont2mcont, cont-intro]*:

**shows** *mcont-restrict-spmf: mcont lub-spmf (ord-spmf (=)) lub-spmf (ord-spmf (=)) (λp. restrict-spmf p A)*  
⟨proof⟩

**lemma** *ord-spmf-case-option*:  $\text{ord-spmf } R \text{ (case } x \text{ of None } \Rightarrow a \mid \text{Some } y \Rightarrow b \ y)$   
 $\text{(case } x \text{ of None } \Rightarrow a' \mid \text{Some } y \Rightarrow b' \ y)$   
**if**  $\text{ord-spmf } R \ a \ a' \wedge y. \text{ord-spmf } R \ (b \ y) \ (b' \ y)$  *<proof>*

**lemma** *ord-spmf-map-spmfI*:  $\text{ord-spmf } (=) \ (\text{map-spmf } f \ p) \ (\text{map-spmf } f \ q)$  **if**  
 $\text{ord-spmf } (=) \ p \ q$   
*<proof>*

### 1.1.1 Conditional probabilities

**lemma** *mk-lossless-cond-spmf [simp]*:  $\text{mk-lossless } (\text{cond-spmf } p \ A) = \text{cond-spmf } p \ A$   
*<proof>*

**context**

**fixes**  $p :: 'a \ \text{pmf}$   
**and**  $f :: 'a \Rightarrow 'b \ \text{pmf}$   
**and**  $A :: 'b \ \text{set}$   
**and**  $F :: 'a \Rightarrow \text{real}$

**defines**  $F \equiv \lambda x. \text{pmf } p \ x * \text{measure } (\text{measure-pmf } (f \ x)) \ A / \text{measure } (\text{measure-pmf } (\text{bind-pmf } p \ f)) \ A$

**begin**

**definition** *cond-bind-pmf* ::  $'a \ \text{pmf}$  **where**  $\text{cond-bind-pmf} = \text{embed-pmf } F$

**lemma** *cond-bind-pmf-nonneg*:  $F \ x \geq 0$   
*<proof>*

**context assumes** *defined*:  $A \cap (\bigcup x \in \text{set-pmf } p. \text{set-pmf } (f \ x)) \neq \{\}$  **begin**

**private lemma** *nonzero*:  $\text{measure } (\text{measure-pmf } (\text{bind-pmf } p \ f)) \ A > 0$   
*<proof>*

**lemma** *cond-bind-pmf-prob*:  $(\int^+ x. F \ x \ \partial \text{count-space } \text{UNIV}) = 1$   
*<proof>*

**lemma** *pmf-cond-bind-pmf*:  $\text{pmf } \text{cond-bind-pmf } x = F \ x$   
*<proof>*

**lemma** *set-cond-bind-pmf*:  $\text{set-pmf } \text{cond-bind-pmf} = \{x \in \text{set-pmf } p. \text{set-pmf } (f \ x) \cap A \neq \{\}\}$   
*<proof>*

**lemma** *cond-bind-pmf*:  $\text{cond-pmf } (\text{bind-pmf } p \ f) \ A = \text{bind-pmf } \text{cond-bind-pmf} \ (\lambda x. \text{cond-pmf } (f \ x) \ A)$   
**(is ?lhs = ?rhs)**  
*<proof>*

**end**

**end**

**lemma** *cond-spmf-try1*:

$cond\text{-}spmf\ (try\text{-}spmf\ p\ q)\ A = cond\text{-}spmf\ p\ A$  **if**  $set\text{-}spmf\ q \cap A = \{\}$   
*<proof>*

**lemma** *cond-spmf-cong*:  $cond\text{-}spmf\ p\ A = cond\text{-}spmf\ p\ B$  **if**  $A \cap set\text{-}spmf\ p = B \cap set\text{-}spmf\ p$

*<proof>*

**lemma** *cond-spmf-pair-spmf*:

$cond\text{-}spmf\ (pair\text{-}spmf\ p\ q)\ (A \times B) = pair\text{-}spmf\ (cond\text{-}spmf\ p\ A)\ (cond\text{-}spmf\ q\ B)$  **(is ?lhs = ?rhs)**  
*<proof>*

**lemma** *cond-spmf-pair-spmf1*:

$cond\text{-}spmf\text{-}fst\ (map\text{-}spmf\ (\lambda((x, s'), y). (f\ x, s', y))\ (pair\text{-}spmf\ p\ q))\ x =$   
 $pair\text{-}spmf\ (cond\text{-}spmf\text{-}fst\ (map\text{-}spmf\ (\lambda(x, s'). (f\ x, s'))\ p)\ x)\ q$  **(is ?lhs = ?rhs)**  
**if** *lossless-spmf*  $q$   
*<proof>*

**lemma** *try-cond-spmf*:  $try\text{-}spmf\ (cond\text{-}spmf\ p\ A)\ q = (if\ set\text{-}spmf\ p \cap A \neq \{\}$  *then*  
 $cond\text{-}spmf\ p\ A$  *else*  $q)$

*<proof>*

**lemma** *cond-spmf-try2*:

$cond\text{-}spmf\ (try\text{-}spmf\ p\ q)\ A = (if\ lossless\text{-}spmf\ p$  *then*  $return\text{-}pmf\ None$  *else*  
 $cond\text{-}spmf\ q\ A)$  **if**  $set\text{-}spmf\ p \cap A = \{\}$   
*<proof>*

**definition** *cond-bind-spmf* ::  $'a\ spmf \Rightarrow ('a \Rightarrow 'b\ spmf) \Rightarrow 'b\ set \Rightarrow 'a\ spmf$  **where**

$cond\text{-}bind\text{-}spmf\ p\ f\ A =$   
 $(if\ \exists x \in set\text{-}spmf\ p. set\text{-}spmf\ (f\ x) \cap A \neq \{\}$  *then*  
 $cond\text{-}bind\text{-}pmf\ p\ (\lambda x. case\ x\ of\ None \Rightarrow return\text{-}pmf\ None \mid Some\ x \Rightarrow f\ x)$   
 $(Some\ 'A)$   
*else*  $return\text{-}pmf\ None)$

**context begin**

**private lemma** *defined*:  $\llbracket y \in set\text{-}spmf\ (f\ x); y \in A; x \in set\text{-}spmf\ p \rrbracket$

$\implies Some\ 'A \cap (\bigcup x \in set\text{-}pmf\ p. set\text{-}pmf\ (case\ x\ of\ None \Rightarrow return\text{-}pmf\ None \mid$   
 $Some\ x \Rightarrow f\ x)) \neq \{\}$

*<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]:

$set-spmf (cond-bind-spmf p f A) = \{x \in set-spmf p. set-spmf (f x) \cap A \neq \{\}\}$   
{proof}

**lemma** *cond-bind-spmf*:  $cond-spmf (bind-spmf p f) A = bind-spmf (cond-bind-spmf p f A) (\lambda x. cond-spmf (f x) A)$   
{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*:

$cond-spmf-fst (map-spmf (\lambda(x, y). (f x, g x y)) p) (f x) = map-spmf (g x)$   
 $(cond-spmf-fst p x)$   
**if** *inj-on f* (*insert x (fst ' set-spmf p)*)  
{proof}

**lemma** *cond-spmf-fst-map-prod-inj*:

$cond-spmf-fst (map-spmf (\lambda(x, y). (f x, g x y)) p) (f x) = map-spmf (g x)$   
 $(cond-spmf-fst p x)$   
**if** *inj f*  
{proof}

**definition** *cond-bind-spmf-fst* ::  $'a\ spmf \Rightarrow ('a \Rightarrow 'b\ spmf) \Rightarrow 'b \Rightarrow 'a\ spmf$  **where**  
 $cond-bind-spmf-fst p f x = cond-bind-spmf p (map-spmf (\lambda b. (b, ()))) \circ f) (\{x\} \times UNIV)$

**lemma** *cond-bind-spmf-fst-map-spmf-fst*:

$cond-bind-spmf-fst p (map-spmf fst \circ f) x = cond-bind-spmf p f (\{x\} \times UNIV)$   
(**is** ?lhs = ?rhs)  
{proof}

**lemma** *cond-spmf-fst-bind*:  $cond-spmf-fst (bind-spmf p f) x =$

$bind-spmf (cond-bind-spmf-fst p (map-spmf fst \circ f) x) (\lambda y. cond-spmf-fst (f y) x)$   
{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]:

$$\text{set-spmf } (\text{cond-bind-spmf-fst } p \ f \ x) = \{y \in \text{set-spmf } p. \ x \in \text{set-spmf } (f \ y)\}$$

*<proof>*

**lemma** *map-cond-spmf-fst*:  $\text{map-spmf } f \ (\text{cond-spmf-fst } p \ x) = \text{cond-spmf-fst } (\text{map-spmf } (\text{apsnd } f) \ p) \ x$

*<proof>*

**lemma** *cond-spmf-fst-try1*:

$$\text{cond-spmf-fst } (\text{try-spmf } p \ q) \ x = \text{cond-spmf-fst } p \ x \ \mathbf{if} \ x \notin \text{fst } \text{'set-spmf } q$$

*<proof>*

**lemma** *cond-spmf-fst-try2*:

$$\text{cond-spmf-fst } (\text{try-spmf } p \ q) \ x = (\text{if } \text{lossless-spmf } p \ \text{then } \text{return-pmf } \text{None} \ \text{else } \text{cond-spmf-fst } q \ x) \ \mathbf{if} \ x \notin \text{fst } \text{'set-spmf } p$$

*<proof>*

**lemma** *cond-spmf-fst-map-inj*:

$$\text{cond-spmf-fst } (\text{map-spmf } (\text{apfst } f) \ p) \ (f \ x) = \text{cond-spmf-fst } p \ x \ \mathbf{if} \ \text{inj } f$$

*<proof>*

**lemma** *cond-spmf-fst-pair-spmf1*:

$$\text{cond-spmf-fst } (\text{map-spmf } (\lambda(x, y). \ (f \ x, \ g \ x \ y)) \ (\text{pair-spmf } p \ q)) \ a = \text{bind-spmf } (\text{cond-spmf-fst } (\text{map-spmf } (\lambda x. \ (f \ x, \ x)) \ p) \ a) \ (\lambda x. \ \text{map-spmf } (g \ x) \ (\text{mk-lossless } q)) \ (\mathbf{is} \ ?lhs = ?rhs)$$

*<proof>*

**lemma** *cond-spmf-fst-return-spmf'*:

$$\text{cond-spmf-fst } (\text{return-spmf } (x, \ y)) \ z = (\text{if } x = z \ \text{then } \text{return-spmf } y \ \text{else } \text{return-pmf } \text{None})$$

*<proof>*

## 2 Material for CryptHOL

**lemma** *left-gpv-lift-spmf* [simp]:  $\text{left-gpv } (\text{lift-spmf } p) = \text{lift-spmf } p$

*<proof>*

**lemma** *right-gpv-lift-spmf* [simp]:  $\text{right-gpv } (\text{lift-spmf } p) = \text{lift-spmf } p$

*<proof>*

**lemma** *map'-lift-spmf*:  $\text{map-gpv}' \ f \ g \ h \ (\text{lift-spmf } p) = \text{lift-spmf } (\text{map-spmf } f \ p)$

*<proof>*

**lemma** *in-set-sample-uniform* [simp]:  $x \in \text{set-spmf } (\text{sample-uniform } n) \iff x < n$

*<proof>*

**lemma** (in *cyclic-group*) *inj-on-generator-iff* [simp]:  $\llbracket x < \text{order } G; \ y < \text{order } G \rrbracket \implies \mathbf{g} \ [\wedge] \ x = \mathbf{g} \ [\wedge] \ y \iff x = y$

$\langle \text{proof} \rangle$

**lemma** *map- $\mathcal{I}$ -bot* [*simp*]:  $\text{map-}\mathcal{I} f g \perp = \perp$   
 $\langle \text{proof} \rangle$

**lemma** *map- $\mathcal{I}$ -Inr-plus* [*simp*]:  $\text{map-}\mathcal{I} \text{Inr} f (\mathcal{I}1 \oplus_{\mathcal{I}} \mathcal{I}2) = \text{map-}\mathcal{I} \text{id} (f \circ \text{Inr}) \mathcal{I}2$   
 $\langle \text{proof} \rangle$

**lemma** *interaction-bound-map-gpv'-le*:

**defines**  $\text{ib} \equiv \text{interaction-bound}$

**shows**  $\text{interaction-bound consider} (\text{map-gpv}' f g h \text{gpv}) \leq \text{ib} (\text{consider} \circ g) \text{gpv}$   
 $\langle \text{proof} \rangle$

**lemma** *interaction-bounded-by-map-gpv'* [*interaction-bound*]:

**assumes**  $\text{interaction-bounded-by} (\text{consider} \circ g) \text{gpv} n$

**shows**  $\text{interaction-bounded-by consider} (\text{map-gpv}' f g h \text{gpv}) n$

$\langle \text{proof} \rangle$

**lemma** *map-gpv'-bind-gpv*:

$\text{map-gpv}' f g h (\text{bind-gpv} \text{gpv} F) = \text{bind-gpv} (\text{map-gpv}' \text{id} g h \text{gpv}) (\lambda x. \text{map-gpv}' f g h (F x))$

$\langle \text{proof} \rangle$

**lemma** *exec-gpv-map-gpv'*:

$\text{exec-gpv} \text{callee} (\text{map-gpv}' f g h \text{gpv}) s =$

$\text{map-spmf} (\text{map-prod} f \text{id}) (\text{exec-gpv} (\text{map-fun} \text{id} (\text{map-fun} g (\text{map-spmf} (\text{map-prod} h \text{id})))) \text{callee} \text{gpv} s)$

$\langle \text{proof} \rangle$

**lemma** *colossless-gpv-sub-gpvs*:

**assumes**  $\text{colossless-gpv} \mathcal{I} \text{gpv} \text{gpv}' \in \text{sub-gpvs} \mathcal{I} \text{gpv}$

**shows**  $\text{colossless-gpv} \mathcal{I} \text{gpv}'$

$\langle \text{proof} \rangle$

**lemma** *pfinite-gpv-sub-gpvs*:

**assumes**  $\text{pfinite-gpv} \mathcal{I} \text{gpv} \text{gpv}' \in \text{sub-gpvs} \mathcal{I} \text{gpv} \mathcal{I} \vdash g \text{gpv} \checkmark$

**shows**  $\text{pfinite-gpv} \mathcal{I} \text{gpv}'$

$\langle \text{proof} \rangle$

**lemma** *pfinite-gpv-id-oracle* [*simp*]:  $\text{pfinite-gpv} \mathcal{I} (\text{id-oracle} s x)$  **if**  $x \in \text{outs-}\mathcal{I} \mathcal{I}$

$\langle \text{proof} \rangle$

## 2.1 *try-gpv*

**lemma** *plossless-gpv-try-gpvI*:

**assumes**  $\text{pfinite-gpv} \mathcal{I} \text{gpv}$

**and**  $\neg \text{colossless-gpv} \mathcal{I} \text{gpv} \implies \text{plossless-gpv} \mathcal{I} \text{gpv}'$

**shows**  $\text{plossless-gpv} \mathcal{I} (\text{TRY} \text{gpv} \text{ELSE} \text{gpv}')$

$\langle \text{proof} \rangle$

**lemma** *WT-gpv-try-gpvI* [*WT-intro*]:

**assumes**  $\mathcal{I} \vdash_g \text{gpv} \checkmark$   
**and**  $\neg \text{colossless-gpv } \mathcal{I} \text{ gpv} \implies \mathcal{I} \vdash_g \text{gpv}' \checkmark$   
**shows**  $\mathcal{I} \vdash_g \text{try-gpv } \text{gpv} \text{ gpv}' \checkmark$   
 $\langle \text{proof} \rangle$

**lemma** (in *callee-invariant-on*) *exec-gpv-try-gpv*:

**fixes** *exec-gpv1*  
**defines**  $\text{exec-gpv1} \equiv \text{exec-gpv}$   
**assumes** *WT*:  $\mathcal{I} \vdash_g \text{gpv} \checkmark$   
**and** *pfinite*:  $\text{pfinite-gpv } \mathcal{I} \text{ gpv}$   
**and** *I*:  $I \text{ s}$   
**and** *f*:  $\bigwedge s. I \text{ s} \implies f(x, s) = z$   
**and** *lossless*:  $\bigwedge s x. \llbracket x \in \text{outs-}\mathcal{I} \text{ } \mathcal{I}; I \text{ s} \rrbracket \implies \text{lossless-spmf } (\text{callee } s \ x)$   
**shows**  $\text{map-spmf } f \ (\text{exec-gpv } \text{callee } (\text{try-gpv } \text{gpv} \ (\text{Done } x)) \ s) =$   
 $\text{try-spmf } (\text{map-spmf } f \ (\text{exec-gpv1 } \text{callee } \text{gpv} \ s)) \ (\text{return-spmf } z)$   
**(is ?lhs = ?rhs)**  
 $\langle \text{proof} \rangle$

**lemma** *try-gpv-bind-gen-lossless'*: — generalises *gen-lossless-gpv ?b I-full ?gpv*  $\implies$   
 $\text{TRY } ?\text{gpv} \ggg ?f \ \text{ELSE } ?\text{gpv}' = ?\text{gpv} \ggg (\lambda x. \text{TRY } ?f \ x \ \text{ELSE } ?\text{gpv}')$

**assumes** *lossless*:  $\text{gen-lossless-gpv } b \ \mathcal{I} \ \text{gpv}$   
**and** *WT1*:  $\mathcal{I} \vdash_g \text{gpv} \checkmark$   
**and** *WT2*:  $\mathcal{I} \vdash_g \text{gpv}' \checkmark$   
**and** *WTf*:  $\bigwedge x. x \in \text{results-gpv } \mathcal{I} \ \text{gpv} \implies \mathcal{I} \vdash_g f \ x \checkmark$   
**shows**  $\text{eq-}\mathcal{I}\text{-gpv } (=) \ \mathcal{I} \ (\text{TRY } \text{bind-gpv } \text{gpv} \ f \ \text{ELSE } \text{gpv}') \ (\text{bind-gpv } \text{gpv} \ (\lambda x. \text{TRY } f \ x \ \text{ELSE } \text{gpv}'))$   
 $\langle \text{proof} \rangle$

**lemmas**  $\text{try-gpv-bind-lossless}' = \text{try-gpv-bind-gen-lossless}'[\mathbf{where } b = \text{False}]$   
**and**  $\text{try-gpv-bind-colossless}' = \text{try-gpv-bind-gen-lossless}'[\mathbf{where } b = \text{True}]$

**lemma** *try-gpv-bind-gpv*:

$\text{try-gpv } (\text{bind-gpv } \text{gpv} \ f) \ \text{gpv}' =$   
 $\text{bind-gpv } (\text{try-gpv } (\text{map-gpv } \text{Some } \text{id } \text{gpv}) \ (\text{Done } \text{None})) \ (\lambda x. \text{case } x \ \text{of } \text{None} \Rightarrow$   
 $\text{gpv}' \mid \text{Some } x' \Rightarrow \text{try-gpv } (f \ x') \ \text{gpv}')$   
 $\langle \text{proof} \rangle$

**lemma** *bind-gpv-try-gpv-map-Some*:

$\text{bind-gpv } (\text{try-gpv } (\text{map-gpv } \text{Some } \text{id } \text{gpv}) \ (\text{Done } \text{None})) \ (\lambda x. \text{case } x \ \text{of } \text{None} \Rightarrow$   
 $\text{Fail} \mid \text{Some } y \Rightarrow f \ y) =$   
 $\text{bind-gpv } \text{gpv} \ f$   
 $\langle \text{proof} \rangle$

**lemma** *try-gpv-left-gpv*:

**assumes**  $\mathcal{I} \vdash_g \text{gpv} \checkmark$  **and** *WT2*:  $\mathcal{I} \vdash_g \text{gpv}' \checkmark$   
**shows**  $\text{eq-}\mathcal{I}\text{-gpv } (=) \ (\mathcal{I} \oplus_{\mathcal{I}} \mathcal{I}') \ (\text{try-gpv } (\text{left-gpv } \text{gpv}) \ (\text{left-gpv } \text{gpv}')) \ (\text{left-gpv}$   
 $(\text{try-gpv } \text{gpv} \ \text{gpv}'))$   
 $\langle \text{proof} \rangle$

**lemma** *try-gpv-right-gpv*:

**assumes**  $\mathcal{I}' \vdash_g \text{gpv} \checkmark$  **and** *WT2*:  $\mathcal{I}' \vdash_g \text{gpv}' \checkmark$   
**shows**  $\text{eq-}\mathcal{I}\text{-gpv} (=) (\mathcal{I} \oplus_{\mathcal{I}} \mathcal{I}') (\text{try-gpv} (\text{right-gpv} \text{gpv}) (\text{right-gpv} \text{gpv}')) (\text{right-gpv} (\text{try-gpv} \text{gpv} \text{gpv}'))$   
*<proof>*

**lemma** *bind-try-Done-Fail*:  $\text{bind-gpv} (\text{TRY} \text{gpv} \text{ELSE} \text{Done} \ x) \ f = \text{bind-gpv} \ \text{gpv} \ f$   
**if**  $f \ x = \text{Fail}$

*<proof>*

**lemma** *inline-map-gpv'*:

$\text{inline} \ \text{callee} \ (\text{map-gpv}' \ f \ g \ h \ \text{gpv}) \ s =$   
 $\text{map-gpv} \ (\text{apfst} \ f) \ \text{id} \ (\text{inline} \ (\text{map-fun} \ \text{id} \ (\text{map-fun} \ g \ (\text{map-gpv} \ (\text{apfst} \ h) \ \text{id})))$   
 $\text{callee} \ \text{gpv} \ s)$   
*<proof>*

**lemma** *interaction-bound-try-gpv*:

**fixes** *consider* **defines**  $\text{ib} \equiv \text{interaction-bound} \ \text{consider}$   
**shows**  $\text{interaction-bound} \ \text{consider} \ (\text{try-gpv} \ \text{gpv} \ \text{gpv}') \leq \text{ib} \ \text{gpv} + \text{ib} \ \text{gpv}'$   
*<proof>*

**lemma** *interaction-bounded-by-try-gpv* [*interaction-bound*]:

$\text{interaction-bounded-by} \ \text{consider} \ (\text{try-gpv} \ \text{gpv1} \ \text{gpv2}) \ (\text{bound1} + \text{bound2})$   
**if**  $\text{interaction-bounded-by} \ \text{consider} \ \text{gpv1} \ \text{bound1} \ \text{interaction-bounded-by} \ \text{consider} \ \text{gpv2} \ \text{bound2}$   
*<proof>*

## 2.2 term *gpv-stop*

**lemma** *interaction-bounded-by-gpv-stop* [*interaction-bound*]:

**assumes**  $\text{interaction-bounded-by} \ \text{consider} \ \text{gpv} \ n$   
**shows**  $\text{interaction-bounded-by} \ \text{consider} \ (\text{gpv-stop} \ \text{gpv}) \ n$   
*<proof>*

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

**lift-definition**  $\text{stop-}\mathcal{I} :: ('a, 'b) \ \mathcal{I} \Rightarrow ('a, 'b) \ \text{option} \ \mathcal{I} \ \text{is}$

$\lambda \text{resp} \ x. \ \text{if} \ (\text{resp} \ x = \{\}) \ \text{then} \ \{\} \ \text{else} \ \text{insert} \ \text{None} \ (\text{Some} \ \text{' resp} \ x) \ \langle \text{proof} \rangle$

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

*<proof>*

**lemma** *responses-stop- $\mathcal{I}$*  [*simp*]:

$\text{responses-}\mathcal{I} \ (\text{stop-}\mathcal{I} \ \mathcal{I}) \ x = (\text{if} \ x \in \text{outs-}\mathcal{I} \ \mathcal{I} \ \text{then} \ \text{insert} \ \text{None} \ (\text{Some} \ \text{' responses-}\mathcal{I} \ \mathcal{I} \ x) \ \text{else} \ \{\})$   
*<proof>*

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

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

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

**end**

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

**lemma** *WT-gpv-stop* [simp, WT-intro]:  $stop\text{-}\mathcal{I}\ \mathcal{I} \vdash_g\ gpv\text{-}stop\ gpv\ \checkmark$  **if**  $\mathcal{I} \vdash_g\ gpv\ \checkmark$   
 ⟨proof⟩

**lemma** *expectation-gpv-stop*:  
**fixes** *fail* **and**  $gpv :: ('a, 'b, 'c)\ gpv$   
**assumes**  $WT: \mathcal{I} \vdash_g\ gpv\ \checkmark$   
**and** *fail*:  $fail \leq c$   
**shows**  $expectation\text{-}gpv\ fail\ (stop\text{-}\mathcal{I}\ \mathcal{I})\ (\lambda\cdot. c)\ (gpv\text{-}stop\ gpv) = expectation\text{-}gpv\ fail\ \mathcal{I}\ (\lambda\cdot. c)\ gpv$  (**is** ?lhs = ?rhs)  
 ⟨proof⟩

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

**lemma** *pfinite-gpv-stop* [simp]:  
 $pfinite\text{-}gpv\ (stop\text{-}\mathcal{I}\ \mathcal{I})\ (gpv\text{-}stop\ gpv) \longleftrightarrow pfinite\text{-}gpv\ \mathcal{I}\ gpv$  **if**  $\mathcal{I} \vdash_g\ gpv\ \checkmark$   
 ⟨proof⟩

**lemma** *plossless-gpv-stop* [simp]:  
 $plossless\text{-}gpv\ (stop\text{-}\mathcal{I}\ \mathcal{I})\ (gpv\text{-}stop\ gpv) \longleftrightarrow plossless\text{-}gpv\ \mathcal{I}\ gpv$  **if**  $\mathcal{I} \vdash_g\ gpv\ \checkmark$   
 ⟨proof⟩

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

**lemma** *Some-in-results'-gpv-gpv-stopD*:  $Some\ xy \in results'\text{-}gpv\ (gpv\text{-}stop\ gpv) \implies xy \in results'\text{-}gpv\ gpv$

*<proof>*

### 2.3 term *exception- $\mathcal{I}$*

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

**lemma** *inj-on-OK* [*simp*]: *inj-on OK A*  
*<proof>*

**function** *join-exception* :: *'a exception*  $\Rightarrow$  *'b exception*  $\Rightarrow$  (*'a*  $\times$  *'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*  $\Rightarrow$  (*'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*  $\Rightarrow$  *'a option* **where**  
*option-of-exception Fault* = *None*  
| *option-of-exception (OK x)* = *Some x*

**fun** *exception-of-option* :: *'a option*  $\Rightarrow$  *'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- $\mathcal{I}$*  :: ('a, 'b)  $\mathcal{I} \Rightarrow$  ('a, 'b *exception*)  $\mathcal{I}$  **where**  
*exception- $\mathcal{I}$   $\mathcal{I}$  = map- $\mathcal{I}$  id exception-of-option (stop- $\mathcal{I}$   $\mathcal{I}$ )*

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

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

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

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

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

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

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

## 2.4 inline

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

**context**

**fixes** *gpv* :: ('a, 'call, 'ret) *gpv*

**assumes** *gpv*: *plossless-gpv  $\mathcal{I}$  gpv  $\mathcal{I} \vdash g$  gpv  $\surd$*

**begin**

**lemma** *lossless-spmf-inline1*:

**assumes** *lossless*:  $\bigwedge s x. \llbracket x \in \text{outs-}\mathcal{I} \ \mathcal{I}; I \ s \rrbracket \implies \text{lossless-spmf (the-gpv (callee } s \ x))$

**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**  $WT: \mathcal{I} \vdash_g gpv \checkmark$   
**and** *pfinite*: *pfinite-gpv*  $\mathcal{I} gpv$   
**and**  $f: \bigwedge s. I s \implies f(x, s) = z$   
**and** *lossless*:  $\bigwedge s x. \llbracket x \in outs\text{-}\mathcal{I} \ \mathcal{I}; I s \rrbracket \implies colossless-gpv \ \mathcal{I}' (callee \ s \ x)$   
**and**  $I: I s$   
**shows**  $map\text{-}spmf (map\text{-}sum \ f \ id) (inline1 \ callee \ (try\text{-}gpv \ gpv \ (Done \ x)) \ s) =$   
 $try\text{-}spmf (map\text{-}spmf (map\text{-}sum \ f \ (\lambda(out, c, rpv). (out, c, \lambda input. try\text{-}gpv (rpv$   
 $input) (Done \ x)))) (inline1' \ callee \ gpv \ s)) (return\text{-}spmf (Inl \ z))$   
**(is** *?lhs = ?rhs*)  
 ⟨*proof*⟩

**lemma** (**in** *raw-converter-invariant*) *inline-try-gpv*:

**assumes**  $WT: \mathcal{I} \vdash_g gpv \checkmark$   
**and** *pfinite*: *pfinite-gpv*  $\mathcal{I} gpv$   
**and**  $f: \bigwedge s. I s \implies f(x, s) = z$   
**and** *lossless*:  $\bigwedge s x. \llbracket x \in outs\text{-}\mathcal{I} \ \mathcal{I}; I s \rrbracket \implies colossless-gpv \ \mathcal{I}' (callee \ s \ x)$   
**and**  $I: I s$   
**shows**  $eq\text{-}\mathcal{I}\text{-}gpv (=) \ \mathcal{I}' (map\text{-}gpv \ f \ id (inline \ callee \ (try\text{-}gpv \ gpv \ (Done \ x)) \ s))$   
 $(try\text{-}gpv (map\text{-}gpv \ f \ id (inline \ callee \ gpv \ s)) (Done \ z))$   
**(is** *eq- $\mathcal{I}$ -gpv - - ?lhs ?rhs*)  
 ⟨*proof*⟩

**definition** *cr-prod2* ::  $'a \Rightarrow ('b \Rightarrow 'c \Rightarrow bool) \Rightarrow 'b \Rightarrow 'a \times 'c \Rightarrow bool$  **where**  
 $cr\text{-}prod2 \ x \ A = (\lambda b \ (a, c). A \ b \ c \wedge x = a)$

**lemma** *cr-prod2-simps* [*simp*]:  $cr\text{-}prod2 \ x \ A \ a \ (b, c) \longleftrightarrow A \ a \ c \wedge x = b$   
 ⟨*proof*⟩

**lemma** *cr-prod2I*:  $A \ a \ b \implies cr\text{-}prod2 \ x \ A \ a \ (x, b)$  ⟨*proof*⟩

**lemma** *cr-prod2-Grp*:  $cr\text{-}prod2 \ x \ (BNF\text{-}Def.\text{Grp} \ A \ f) = BNF\text{-}Def.\text{Grp} \ A \ (\lambda b. (x, f \ b))$   
 ⟨*proof*⟩

**lemma** *extend-state-oracle-transfer'*: **includes** *lifting-syntax* **shows**

$((S \ ==\!>\!> \ C \ ==\!>\!> \ rel\text{-}spmf \ (rel\text{-}prod \ R \ S)) \ ==\!>\!> \ cr\text{-}prod2 \ s \ S \ ==\!>\!> \ C$   
 $\ ==\!>\!> \ rel\text{-}spmf \ (rel\text{-}prod \ R \ (cr\text{-}prod2 \ s \ S))) (\lambda oracle. oracle) \ extend\text{-}state\text{-}oracle$

$\langle \text{proof} \rangle$

**lemma** *exec-gpv-extend-state-oracle*:

$\text{exec-gpv} (\text{extend-state-oracle } \text{callee}) \text{ gpv } (s, s') =$   
 $\text{map-spmf } (\lambda(x, s''). (x, (s, s''))) (\text{exec-gpv } \text{callee } \text{gpv } s')$   
 $\langle \text{proof} \rangle$

### 3 Material for Constructive Crypto

**lemma** *WT-resource-I-uniform-UNIV* [*simp*]:  $\mathcal{I}$ -uniform  $A$   $UNIV \vdash_{\text{res}} \text{res} \checkmark$   
 $\langle \text{proof} \rangle$

**lemma** *WT-converter-of-callee-invar*:

**assumes**  $WT: \bigwedge s q. \llbracket q \in \text{outs-}\mathcal{I} \ \mathcal{I}; I s \rrbracket \implies \mathcal{I}' \vdash_g \text{callee } s \ q \ \checkmark$   
**and**  $\text{res}: \bigwedge s q r s'. \llbracket (r, s') \in \text{results-gpv } \mathcal{I}' (\text{callee } s \ q); q \in \text{outs-}\mathcal{I} \ \mathcal{I}; I s \rrbracket$   
 $\implies r \in \text{responses-}\mathcal{I} \ \mathcal{I} \ q \wedge I s'$   
**and**  $I: I s$   
**shows**  $\mathcal{I}, \mathcal{I}' \vdash_C \text{converter-of-callee } \text{callee } s \ \checkmark$   
 $\langle \text{proof} \rangle$

**lemma** *eq-I-gpv-eq-OO*:

**assumes**  $\text{eq-}\mathcal{I}\text{-gpv } (=) \ \mathcal{I} \ \text{gpv} \ \text{gpv}' \ \text{eq-}\mathcal{I}\text{-gpv } A \ \mathcal{I} \ \text{gpv}' \ \text{gpv}''$   
**shows**  $\text{eq-}\mathcal{I}\text{-gpv } A \ \mathcal{I} \ \text{gpv} \ \text{gpv}''$   
 $\langle \text{proof} \rangle$

**lemma** *eq-I-gpv-eq-OO2*:

**assumes**  $\text{eq-}\mathcal{I}\text{-gpv } (=) \ \mathcal{I} \ \text{gpv}'' \ \text{gpv}' \ \text{eq-}\mathcal{I}\text{-gpv } A \ \mathcal{I} \ \text{gpv} \ \text{gpv}'$   
**shows**  $\text{eq-}\mathcal{I}\text{-gpv } A \ \mathcal{I} \ \text{gpv} \ \text{gpv}''$   
 $\langle \text{proof} \rangle$

**lemma** *eq-I-gpv-try-gpv-cong*:

**assumes**  $\text{eq-}\mathcal{I}\text{-gpv } A \ \mathcal{I} \ \text{gpv}1 \ \text{gpv}1'$   
**and**  $\text{eq-}\mathcal{I}\text{-gpv } A \ \mathcal{I} \ \text{gpv}2 \ \text{gpv}2'$   
**shows**  $\text{eq-}\mathcal{I}\text{-gpv } A \ \mathcal{I} \ (\text{try-gpv } \text{gpv}1 \ \text{gpv}2) \ (\text{try-gpv } \text{gpv}1' \ \text{gpv}2')$   
 $\langle \text{proof} \rangle$

**lemma** *eq-I-gpv-map-gpv'*:

**assumes**  $\text{eq-}\mathcal{I}\text{-gpv } (\text{BNF-Def.vimage2p } f \ f' \ A) \ (\text{map-}\mathcal{I} \ g \ h \ \mathcal{I}) \ \text{gpv}1 \ \text{gpv}2$   
**shows**  $\text{eq-}\mathcal{I}\text{-gpv } A \ \mathcal{I} \ (\text{map-gpv}' \ f \ g \ h \ \text{gpv}1) \ (\text{map-gpv}' \ f' \ g \ h \ \text{gpv}2)$   
 $\langle \text{proof} \rangle$

**lemma** *eq-I-converter-map-converter*:

**assumes**  $\text{map-}\mathcal{I} \ (\text{inv-into } UNIV \ f) \ (\text{inv-into } UNIV \ g) \ \mathcal{I}, \ \text{map-}\mathcal{I} \ f' \ g' \ \mathcal{I}' \vdash_C \text{conv}1$   
 $\sim \text{conv}2$   
**and**  $\text{inj } f \ \text{surj } g$   
**shows**  $\mathcal{I}, \mathcal{I}' \vdash_C \text{map-converter } f \ g \ f' \ g' \ \text{conv}1 \sim \text{map-converter } f \ g \ f' \ g' \ \text{conv}2$   
 $\langle \text{proof} \rangle$

**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*]:  $\mathcal{I} \vdash_{\text{res}} \text{fail-resource} \checkmark$   
⟨*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*]:  
 $\mathcal{I} \vdash_{\text{res}} \text{const-resource } y \checkmark \iff (\forall x \in \text{outs-}\mathcal{I} \ \mathcal{I}. y \in \text{responses-}\mathcal{I} \ \mathcal{I} \ x) \text{ (is ?lhs } \iff \text{ ?rhs)}$   
⟨*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* =

( $\lambda x. \text{map-gpv } (\text{map-prod id } (\lambda \text{conv. case conv of None} \Rightarrow \text{merge-exception-converter} \mid \text{Some conv}' \Rightarrow \text{conv}')) \text{id } ($

*Pause* *x* ( $\lambda y. \text{Done } (\text{case merge-exception } y \text{ of Fault} \Rightarrow (\text{Fault, Some } (\text{const-converter } \text{Fault}))$

$\mid \text{OK } y' \Rightarrow (\text{OK } y', \text{None}))))$ )

**lemma** *merge-exception-converter.sel* [*simp*]:

*run-converter* *merge-exception-converter* *x* =

*Pause* *x* ( $\lambda y. \text{Done } (\text{case merge-exception } y \text{ of Fault} \Rightarrow (\text{Fault, const-converter } \text{Fault}) \mid \text{OK } y' \Rightarrow (\text{OK } y', \text{merge-exception-converter}))$ )

⟨*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*]:

$\mathcal{I}, \mathcal{I}' \vdash_C \text{const-converter } x \checkmark$  **if**  $\forall q \in \text{outs-}\mathcal{I} \mathcal{I}. x \in \text{responses-}\mathcal{I} \mathcal{I} q$

⟨*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- $\mathcal{I}$*   $\mathcal{I}2' \vdash_C \text{merge-exception-converter}$

$\checkmark$   
⟨*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*) ( $\lambda(x, conv')$ . *case* *x* of *None*  $\Rightarrow$  *Fail* | *Some* *x'*  $\Rightarrow$  *Done* (*x*, *conv'*)) =  
 $bind\_gpv$  (*left-gpv* (*map-gpv'* *id id option-of-exception* (*gpv-stop gpv*))) ( $\lambda x$ . *case* *x* of *None*  $\Rightarrow$  *Fail* | *Some* *x'*  $\Rightarrow$  *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*) ( $\lambda(x, conv')$ . *case* *x* of *None*  $\Rightarrow$  *Fail* | *Some* *x'*  $\Rightarrow$  *Done* (*x*, *conv'*)) =  
 $bind\_gpv$  (*right-gpv* (*map-gpv'* *id id option-of-exception* (*gpv-stop gpv*))) ( $\lambda x$ . *case* *x* of *None*  $\Rightarrow$  *Fail* | *Some* *x'*  $\Rightarrow$  *Done* (*x*, *merge-exception-converter*))  
 ⟨*proof*⟩

### 3.1 Constructive-Cryptography.Wiring

**abbreviation** (*input*)

$id\_wiring :: ('a, 'b, 'a, 'b) wiring \langle 1_w \rangle$

**where**

$id\_wiring \equiv (id, id)$

**definition**

$swap\_lassocr_w :: ('a + 'b + 'c, 'd + 'e + 'f, 'b + 'a + 'c, 'e + 'd + 'f) wiring$

**where**

$swap\_lassocr_w \equiv rassocl_w \circ_w ((swap_w \mid_w 1_w) \circ_w lassocr_w)$

**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**

$parallel\_wiring_w \equiv lassocr_w \circ_w ((1_w \mid_w swap\_lassocr_w) \circ_w rassocl_w)$

**schematic-goal**

$wiring\_parallel\_wiring[wiring\_intro]: wiring \ ?I1 \ ?I2 parallel\_wiring parallel\_wiring_w$   
 ⟨*proof*⟩

**lemma** *lassocr-inverse*:  $rassocl_C \odot lassocr_C = 1_C$

⟨*proof*⟩

**lemma** *rassocl-inverse*:  $lassocr_C \odot rassocl_C = 1_C$

⟨*proof*⟩

**lemma** *swap-sum-swap-sum* [*simp*]:  $\text{swap-sum} (\text{swap-sum } x) = x$   
(*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*]:  $\text{lassocr-swap-sum} (\text{lassocr-swap-sum } x) = x$   
(*proof*)

**lemma** *inv-lassocr-swap-sum* [*simp*]: *inv-into UNIV lassocr-swap-sum = lassocr-swap-sum*  
(*proof*)

**lemma** *swap-inverse*:  $\text{swap}_C \odot \text{swap}_C = 1_C$   
(*proof*)

**lemma** *swap-lassocr-inverse*:  $\mathcal{I}1 \oplus_{\mathcal{I}} (\mathcal{I}2 \oplus_{\mathcal{I}} \mathcal{I}3), \mathcal{I}1 \oplus_{\mathcal{I}} (\mathcal{I}2 \oplus_{\mathcal{I}} \mathcal{I}3) \vdash_C \text{swap-lassocr} \odot \text{swap-lassocr} \sim 1_C$   
(**is** ? $\mathcal{I}$ , -  $\vdash_C$  ?*lhs*  $\sim$  -)  
(*proof*)

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

(is ? $\mathcal{I}$ ,  $- \vdash_C ?lhs \sim -$ )  
 <proof>

**definition**

*attach-wiring-right* ::  
 ('a, 'b, 'c, 'd) wiring  $\Rightarrow$   
 ('s  $\Rightarrow$  'e  $\Rightarrow$  ('f  $\times$  's, 'a, 'b) gpv)  $\Rightarrow$  ('s  $\Rightarrow$  'e  $\Rightarrow$  ('f  $\times$  's, 'c, 'd) gpv)

**where**

*attach-wiring-right* =  $(\lambda(f, g). \text{map-fun id (map-fun id (map-gpv' id f g))})$

**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  $\mathcal{I}2 \mathcal{I}3 \text{ conv } w$

**and** WT:  $\mathcal{I}1, \mathcal{I}2 \vdash_C \text{CNV callee } s \checkmark$

**shows**  $\mathcal{I}1, \mathcal{I}3 \vdash_C \text{CNV callee } s \odot \text{conv} \sim \text{CNV (attach-wiring-right } w \text{ callee) } s$

<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)  $\mathcal{I} \Rightarrow$  ('c, 'd)  $\mathcal{I} \Rightarrow$  ('a, 'b, 'c, 'd) *converter*  
 $\Rightarrow \text{bool}$

**for**  $\mathcal{I} \mathcal{I}'$  **where**

*pfinite-converterI*: *pfinite-converter*  $\mathcal{I} \mathcal{I}' \text{ conv}$  **if**

$\bigwedge a. a \in \text{outs-}\mathcal{I} \mathcal{I} \Rightarrow \text{pfinite-gpv } \mathcal{I}' (\text{run-converter conv } a)$

$\bigwedge a b \text{ conv}'. \llbracket a \in \text{outs-}\mathcal{I} \mathcal{I}; (b, \text{conv}') \in \text{results-gpv } \mathcal{I}' (\text{run-converter conv } a) \rrbracket$   
 $\Rightarrow \text{pfinite-converter } \mathcal{I} \mathcal{I}' \text{ conv}'$

**lemma** *pfinite-converter-coinduct*[consumes 1, case-names *pfinite-converter*, case-conclusion *pfinite-converter pfinite step, coinduct pred: pfinite-converter*]:

**assumes**  $X \text{ conv}$

**and** *step*:  $\bigwedge \text{conv } a. \llbracket X \text{ conv}; a \in \text{outs-}\mathcal{I} \mathcal{I} \rrbracket \Rightarrow \text{pfinite-gpv } \mathcal{I}' (\text{run-converter conv } a) \wedge$

$(\forall (b, \text{conv}') \in \text{results-gpv } \mathcal{I}' (\text{run-converter conv } a). X \text{ conv}' \vee \text{pfinite-converter } \mathcal{I} \mathcal{I}' \text{ conv}')$

**shows** *pfinite-converter*  $\mathcal{I} \mathcal{I}' \text{ conv}$

<proof>

**lemma** *pfinite-converterD*:  
 $\llbracket \text{pfinite-converter } \mathcal{I} \ \mathcal{I}' \ \text{conv}; a \in \text{outs-}\mathcal{I} \ \mathcal{I} \rrbracket$   
 $\implies \text{pfinite-gpv } \mathcal{I}' \ (\text{run-converter } \text{conv } a) \wedge$   
 $(\forall (b, \text{conv}') \in \text{results-gpv } \mathcal{I}' \ (\text{run-converter } \text{conv } a). \text{pfinite-converter } \mathcal{I} \ \mathcal{I}'$   
 $\text{conv}')$   
 $\langle \text{proof} \rangle$

**lemma** *pfinite-converter-bot1* [*simp*]: *pfinite-converter bot*  $\mathcal{I}$  *conv*  
 $\langle \text{proof} \rangle$

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

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

**lemma** *pfinite-converter-of-callee*:  
**assumes** *step*:  $\bigwedge x \ s. \llbracket x \in \text{outs-}\mathcal{I} \ \mathcal{I}; I \ s \rrbracket \implies \text{pfinite-gpv } \mathcal{I}' \ (\text{callee } s \ x)$   
**and** *I*:  $I \ s$   
**shows** *pfinite-converter*  $\mathcal{I} \ \mathcal{I}' \ (\text{converter-of-callee } \text{callee } s)$   
 $\langle \text{proof} \rangle$

**end**

**lemma** *raw-converter-invariant-run-pfinite-converter*:  
*raw-converter-invariant*  $\mathcal{I} \ \mathcal{I}' \ \text{run-converter} \ (\lambda \text{conv}. \text{pfinite-converter } \mathcal{I} \ \mathcal{I}' \ \text{conv} \wedge$   
 $\mathcal{I}, \mathcal{I}' \vdash_C \text{conv} \checkmark)$   
 $\langle \text{proof} \rangle$

**interpretation** *run-pfinite-converter*: *raw-converter-invariant*  
 $\mathcal{I} \ \mathcal{I}' \ \text{run-converter} \ \lambda \text{conv}. \text{pfinite-converter } \mathcal{I} \ \mathcal{I}' \ \text{conv} \wedge \mathcal{I}, \mathcal{I}' \vdash_C \text{conv} \checkmark$  **for**  $\mathcal{I} \ \mathcal{I}'$   
 $\langle \text{proof} \rangle$

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

**lemma** *pfinite-id-converter* [*pfinite-intro*]: *pfinite-converter*  $\mathcal{I} \ \mathcal{I}$  *id-converter*  
 $\langle \text{proof} \rangle$

**lemma** *pfinite-fail-converter* [*pfinite-intro*]: *pfinite-converter*  $\mathcal{I} \ \mathcal{I}'$  *fail-converter*  
 $\langle \text{proof} \rangle$

**lemma** *pfinite-parallel-converter2* [*pfinite-intro*]:  
*pfinite-converter*  $(\mathcal{I}1 \oplus_{\mathcal{I}} \mathcal{I}2) \ (\mathcal{I}1' \oplus_{\mathcal{I}} \mathcal{I}2') \ (\text{conv}1 \mid_{=} \text{conv}2)$   
**if** *pfinite-converter*  $\mathcal{I}1 \ \mathcal{I}1' \ \text{conv}1$  *pfinite-converter*  $\mathcal{I}2 \ \mathcal{I}2' \ \text{conv}2$   
 $\langle \text{proof} \rangle$

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

**lemma** *expectation-gpv-1-le-inline*:

**defines** *expectation-gpv2*  $\equiv$  *expectation-gpv* 1  $\mathcal{I}'$   
**assumes** *callee*:  $\bigwedge s x. \llbracket x \in \text{outs-}\mathcal{I} \ \mathcal{I}; I \ s \rrbracket \implies \text{pfinite-gpv} \ \mathcal{I}' \ (\text{callee} \ s \ x)$   
**and** *WT-gpv*:  $\mathcal{I} \vdash_g \text{gpv} \ \checkmark$   
**and** *I*:  $I \ s$   
**and** *f-le-1*:  $\bigwedge x. f \ x \leq 1$   
**shows** *expectation-gpv* 1  $\mathcal{I} \ f \ \text{gpv} \leq$  *expectation-gpv2*  $(\lambda(x, s). f \ x)$  (*inline callee gpv s*)  
*<proof>*

**lemma** *pfinite-inline*:

**assumes** *fin*: *pfinite-gpv*  $\mathcal{I} \ \text{gpv}$   
**and** *WT*:  $\mathcal{I} \vdash_g \text{gpv} \ \checkmark$   
**and** *callee*:  $\bigwedge s x. \llbracket x \in \text{outs-}\mathcal{I} \ \mathcal{I}; I \ s \rrbracket \implies \text{pfinite-gpv} \ \mathcal{I}' \ (\text{callee} \ s \ x)$   
**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*  $\odot$  *conv2*)  
**if** *pfinite-converter*  $\mathcal{I}1 \ \mathcal{I}2$  *conv1* *pfinite-converter*  $\mathcal{I}2 \ \mathcal{I}3$  *conv2*  $\mathcal{I}1, \mathcal{I}2 \vdash_C \text{conv1}$   
 $\checkmark \ \mathcal{I}2, \mathcal{I}3 \vdash_C \text{conv2} \ \checkmark$   
*<proof>*

**lemma** *pfinite-map-converter* [*pfinite-intro*]:

*pfinite-converter*  $\mathcal{I} \ \mathcal{I}'$  (*map-converter*  $f \ g \ f' \ g' \ \text{conv}$ ) **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<sub>C</sub>* [*pfinite-intro*]: *pfinite-converter*  $((\mathcal{I}1 \oplus_{\mathcal{I}} \mathcal{I}2) \oplus_{\mathcal{I}} \mathcal{I}3)$   $(\mathcal{I}1$   
 $\oplus_{\mathcal{I}} (\mathcal{I}2 \oplus_{\mathcal{I}} \mathcal{I}3))$  *lassocr<sub>C</sub>*  
*<proof>*

**lemma** *pfinite-rassocl<sub>C</sub>* [*pfinite-intro*]: *pfinite-converter*  $(\mathcal{I}1 \oplus_{\mathcal{I}} (\mathcal{I}2 \oplus_{\mathcal{I}} \mathcal{I}3))$   $((\mathcal{I}1$   
 $\oplus_{\mathcal{I}} \mathcal{I}2) \oplus_{\mathcal{I}} \mathcal{I}3)$  *rassocl<sub>C</sub>*  
*<proof>*

**lemma** *pfinite-swap<sub>C</sub>* [*pfinite-intro*]: *pfinite-converter*  $(\mathcal{I}1 \oplus_{\mathcal{I}} \mathcal{I}2)$   $(\mathcal{I}2 \oplus_{\mathcal{I}} \mathcal{I}1)$   
*swap<sub>C</sub>*  
*<proof>*

**lemma** *pfinite-swap-lassocr* [*pfinite-intro*]: *pfinite-converter*  $(\mathcal{I}1 \oplus_{\mathcal{I}} (\mathcal{I}2 \oplus_{\mathcal{I}} \mathcal{I}3))$   
 $(\mathcal{I}2 \oplus_{\mathcal{I}} (\mathcal{I}1 \oplus_{\mathcal{I}} \mathcal{I}3))$  *swap-lassocr*  
*<proof>*

**lemma** *pfinite-swap-rassocl* [*pfinite-intro*]: *pfinite-converter*  $((\mathcal{I}1 \oplus_{\mathcal{I}} \mathcal{I}2) \oplus_{\mathcal{I}} \mathcal{I}3)$   
 $((\mathcal{I}1 \oplus_{\mathcal{I}} \mathcal{I}3) \oplus_{\mathcal{I}} \mathcal{I}2)$  *swap-rassocl*  
 $\langle \text{proof} \rangle$

**lemma** *pfinite-parallel-wiring* [*pfinite-intro*]:  
*pfinite-converter*  $((\mathcal{I}1 \oplus_{\mathcal{I}} \mathcal{I}2) \oplus_{\mathcal{I}} (\mathcal{I}3 \oplus_{\mathcal{I}} \mathcal{I}4)) ((\mathcal{I}1 \oplus_{\mathcal{I}} \mathcal{I}3) \oplus_{\mathcal{I}} (\mathcal{I}2 \oplus_{\mathcal{I}} \mathcal{I}4))$   
*parallel-wiring*  
 $\langle \text{proof} \rangle$

**lemma** *pfinite-parallel-converter* [*pfinite-intro*]:  
*pfinite-converter*  $(\mathcal{I}1 \oplus_{\mathcal{I}} \mathcal{I}2) \mathcal{I}3$  (*conv1*  $|_{\infty}$  *conv2*)  
**if** *pfinite-converter*  $\mathcal{I}1 \mathcal{I}3$  *conv1* **and** *pfinite-converter*  $\mathcal{I}2 \mathcal{I}3$  *conv2*  
 $\langle \text{proof} \rangle$

**lemma** *pfinite-converter-of-resource* [*simp*, *pfinite-intro*]: *pfinite-converter*  $\mathcal{I}1 \mathcal{I}2$   
(*converter-of-resource* *res*)  
 $\langle \text{proof} \rangle$

### 3.3 colossless converter

**coinductive** *colossless-converter* ::  $(a, b) \mathcal{I} \Rightarrow (c, d) \mathcal{I} \Rightarrow (a, b, c, d)$  *converter*  
 $\Rightarrow \text{bool}$

**for**  $\mathcal{I} \mathcal{I}'$  **where**

*colossless-converterI*:

*colossless-converter*  $\mathcal{I} \mathcal{I}'$  *conv* **if**

$\bigwedge a. a \in \text{outs-}\mathcal{I} \mathcal{I} \implies \text{colossless-gpv } \mathcal{I}' (\text{run-converter } \text{conv } a)$

$\bigwedge a b \text{ conv}'. \llbracket a \in \text{outs-}\mathcal{I} \mathcal{I}; (b, \text{conv}') \in \text{results-gpv } \mathcal{I}' (\text{run-converter } \text{conv } a) \rrbracket$

$\implies \text{colossless-converter } \mathcal{I} \mathcal{I}' \text{ conv}'$

**lemma** *colossless-converter-coinduct*[*consumes 1*, *case-names colossless-converter*,  
*case-conclusion colossless-converter plossless step*, *coinduct pred: colossless-converter*]:

**assumes**  $X \text{ conv}$

**and step:**  $\bigwedge \text{conv } a. \llbracket X \text{ conv}; a \in \text{outs-}\mathcal{I} \mathcal{I} \rrbracket \implies \text{colossless-gpv } \mathcal{I}' (\text{run-converter } \text{conv } a) \wedge$

$(\forall (b, \text{conv}') \in \text{results-gpv } \mathcal{I}' (\text{run-converter } \text{conv } a). X \text{ conv}' \vee \text{colossless-converter } \mathcal{I} \mathcal{I}' \text{ conv}')$

**shows** *colossless-converter*  $\mathcal{I} \mathcal{I}' \text{ conv}$

$\langle \text{proof} \rangle$

**lemma** *colossless-converterD*:

$\llbracket \text{colossless-converter } \mathcal{I} \mathcal{I}' \text{ conv}; a \in \text{outs-}\mathcal{I} \mathcal{I} \rrbracket$

$\implies \text{colossless-gpv } \mathcal{I}' (\text{run-converter } \text{conv } a) \wedge$

$(\forall (b, \text{conv}') \in \text{results-gpv } \mathcal{I}' (\text{run-converter } \text{conv } a). \text{colossless-converter } \mathcal{I} \mathcal{I}' \text{ conv}')$

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

**lemma** *raw-converter-invariant-run-colossless-converter*: *raw-converter-invariant*  
 $\mathcal{I} \mathcal{I}'$  *run-converter*  $(\lambda \text{conv. colossless-converter } \mathcal{I} \mathcal{I}' \text{ conv} \wedge \mathcal{I}, \mathcal{I}' \vdash_C \text{conv} \checkmark)$   
 $\langle \text{proof} \rangle$

**interpretation** *run-colossless-converter*: *raw-converter-invariant*  
 $\mathcal{I} \mathcal{I}'$  *run-converter*  $\lambda \text{conv. colossless-converter } \mathcal{I} \mathcal{I}' \text{ conv} \wedge \mathcal{I}, \mathcal{I}' \vdash_C \text{conv} \checkmark$  **for**  
 $\mathcal{I} \mathcal{I}'$   
 $\langle \text{proof} \rangle$

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

### 3.4 trace equivalence

**lemma** *distinguish-trace-eq*:  
**assumes** *distinguish*:  $\bigwedge \text{distinguisher. } \mathcal{I} \vdash_g \text{distinguisher} \checkmark \implies \text{connect distinguisher } \text{res} = \text{connect distinguisher } \text{res}'$   
**shows** *outs- $\mathcal{I}$*   $\mathcal{I} \vdash_R \text{res} \approx \text{res}'$   
 $\langle \text{proof} \rangle$

**lemma** *attach-trace-eq'*:  
**assumes** *eq*: *outs- $\mathcal{I}$*   $\mathcal{I} \vdash_R \text{res1} \approx \text{res2}$   
**and** *WT1* [*WT-intro*]:  $\mathcal{I} \vdash_{\text{res}} \text{res1} \checkmark$   
**and** *WT2* [*WT-intro*]:  $\mathcal{I} \vdash_{\text{res}} \text{res2} \checkmark$   
**and** *WT-conv* [*WT-intro*]:  $\mathcal{I}', \mathcal{I} \vdash_C \text{conv} \checkmark$   
**shows** *outs- $\mathcal{I}$*   $\mathcal{I}' \vdash_R \text{conv} \triangleright \text{res1} \approx \text{conv} \triangleright \text{res2}$   
 $\langle \text{proof} \rangle$

**lemma** *trace-callee-eq-trans* [*trans*]:  
 $\llbracket \text{trace-callee-eq } \text{callee1 } \text{callee2 } A \ p \ q; \text{trace-callee-eq } \text{callee2 } \text{callee3 } A \ q \ r \rrbracket$   
 $\implies \text{trace-callee-eq } \text{callee1 } \text{callee3 } A \ p \ r$   
 $\langle \text{proof} \rangle$

**lemma** *trace-eq'-parallel-resource*:  
**fixes** *res1* :: ('a, 'b) *resource* **and** *res2* :: ('c, 'd) *resource*  
**assumes** *1*: *trace-eq'*  $A \ \text{res1} \ \text{res1}'$   
**and** *2*: *trace-eq'*  $B \ \text{res2} \ \text{res2}'$   
**shows** *trace-eq'*  $(A \lt;+\gt B) (\text{res1} \parallel \text{res2}) (\text{res1}' \parallel \text{res2}')$   
 $\langle \text{proof} \rangle$

**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*:  $\bigwedge p \ q \ a. \llbracket S \ p \ q; a \in A \rrbracket \implies$   
 $\text{bind-spmf } p \ (\lambda s. \text{map-spmf } \text{fst} (\text{callee1 } s \ a)) = \text{bind-spmf } q \ (\lambda s. \text{map-spmf } \text{fst} (\text{callee2 } s \ a))$   
**and** *sim*:  $\bigwedge p \ q \ a \ \text{res} \ \text{res}' \ b \ s'' \ s'. \llbracket S \ p \ q; a \in A; \text{res} \in \text{set-spmf } p; (b, s'') \in$

$set\text{-}spmf\ (callee1\ res\ a); res' \in set\text{-}spmf\ q; (b, s') \in set\text{-}spmf\ (callee2\ res'\ a) \llbracket$   
 $\implies S\ (cond\text{-}spmf\text{-}fst\ (bind\text{-}spmf\ p\ (\lambda s. callee1\ s\ a))\ b)$   
 $\quad (cond\text{-}spmf\text{-}fst\ (bind\text{-}spmf\ q\ (\lambda s. callee2\ s\ a))\ b)$   
**shows**  $trace\text{-}callee\text{-}eq\ callee1\ callee2\ A\ p\ q$   
 $\langle proof \rangle$

**proposition**  $trace\text{-}callee\text{-}eq\text{-}coinduct\text{-}strong$  [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**  $step: \bigwedge p\ q\ a. \llbracket S\ p\ q; a \in A \rrbracket \implies$   
 $bind\text{-}spmf\ p\ (\lambda s. map\text{-}spmf\ fst\ (callee1\ s\ a)) = bind\text{-}spmf\ q\ (\lambda s. map\text{-}spmf\ fst$   
 $(callee2\ s\ a))$   
**and**  $sim: \bigwedge p\ q\ a\ res\ res'\ b\ s''\ s'. \llbracket S\ p\ q; a \in A; res \in set\text{-}spmf\ p; (b, s') \in$   
 $set\text{-}spmf\ (callee1\ res\ a); res' \in set\text{-}spmf\ q; (b, s') \in set\text{-}spmf\ (callee2\ res'\ a) \rrbracket$   
 $\implies S\ (cond\text{-}spmf\text{-}fst\ (bind\text{-}spmf\ p\ (\lambda s. callee1\ s\ a))\ b)$   
 $\quad (cond\text{-}spmf\text{-}fst\ (bind\text{-}spmf\ q\ (\lambda s. callee2\ s\ a))\ b) \vee$   
 $trace\text{-}callee\text{-}eq\ callee1\ callee2\ A\ (cond\text{-}spmf\text{-}fst\ (bind\text{-}spmf\ p\ (\lambda s. callee1\ s$   
 $a))\ b)\ (cond\text{-}spmf\text{-}fst\ (bind\text{-}spmf\ q\ (\lambda s. callee2\ s\ a))\ b)$   
**shows**  $trace\text{-}callee\text{-}eq\ callee1\ callee2\ A\ p\ q$   
 $\langle proof \rangle$

**lemma**  $trace\text{-}callee\text{-}return\text{-}pmf\text{-}None$  [simp]:

$trace\text{-}callee\text{-}eq\ callee1\ callee2\ A\ (return\text{-}pmf\ None)\ (return\text{-}pmf\ None)$   
 $\langle proof \rangle$

**lemma**  $trace\text{-}callee\text{-}eq\text{-}sym$  [sym]:  $trace\text{-}callee\text{-}eq\ callee1\ callee2\ A\ p\ q \implies trace\text{-}callee\text{-}eq$   
 $callee2\ callee1\ A\ q\ p$

$\langle proof \rangle$

**lemma**  $eq\text{-}resource\text{-}on\text{-}imp\text{-}trace\text{-}eq: A \vdash_R res1 \approx res2$  **if**  $A \vdash_R res1 \sim res2$

$\langle proof \rangle$

**lemma**  $advantage\text{-}nonneg: 0 \leq advantage\ \mathcal{A}\ res1\ res2$

$\langle proof \rangle$

**lemma**  $comp\text{-}converter\text{-}of\text{-}resource\text{-}conv\text{-}parallel\text{-}converter:$

$(converter\text{-}of\text{-}resource\ res\ |_{\infty}\ 1_C) \odot conv = converter\text{-}of\text{-}resource\ res\ |_{\infty}\ conv$

$\langle proof \rangle$

**lemma**  $comp\text{-}converter\text{-}of\text{-}resource\text{-}conv\text{-}parallel\text{-}converter2:$

$(1_C\ |_{\infty}\ converter\text{-}of\text{-}resource\ res) \odot conv = conv\ |_{\infty}\ converter\text{-}of\text{-}resource\ res$

$\langle proof \rangle$

**lemma**  $parallel\text{-}converter\text{-}map\text{-}converter:$

$map\text{-}converter\ f\ g\ f'\ g'\ conv1\ |_{\infty}\ map\text{-}converter\ f''\ g''\ f'\ g'\ conv2 =$

$map\text{-}converter\ (map\text{-}sum\ f\ f'')\ (map\text{-}sum\ g\ g'')\ f'\ g'\ (conv1\ |_{\infty}\ conv2)$

$\langle proof \rangle$

**lemma** *map-converter-parallel-converter-out2*:

$conv1 \mid_{\infty} map\text{-}converter\ f\ g\ id\ id\ conv2 = map\text{-}converter\ (map\text{-}sum\ id\ f)\ (map\text{-}sum\ id\ g)\ id\ id\ (conv1 \mid_{\infty} conv2)$   
 ⟨proof⟩

**lemma** *parallel-converter-assoc2*:

$parallel\text{-}converter\ conv1\ (parallel\text{-}converter\ conv2\ conv3) = map\text{-}converter\ lsumr\ rsuml\ id\ id\ (parallel\text{-}converter\ (parallel\text{-}converter\ conv1\ conv2)\ conv3)$   
 ⟨proof⟩

**lemma** *parallel-converter-of-resource*:

$converter\text{-}of\text{-}resource\ res1 \mid_{\infty} converter\text{-}of\text{-}resource\ res2 = converter\text{-}of\text{-}resource\ (res1 \parallel res2)$   
 ⟨proof⟩

**lemma** *map-Inr-parallel-converter*:

$map\text{-}converter\ Inr\ f\ g\ h\ (conv1 \mid_{\infty} conv2) = map\text{-}converter\ id\ (f \circ Inr)\ g\ h\ conv2$   
 (is ?lhs = ?rhs)  
 ⟨proof⟩

**lemma** *map-Inl-parallel-converter*:

$map\text{-}converter\ Inl\ f\ g\ h\ (conv1 \mid_{\infty} conv2) = map\text{-}converter\ id\ (f \circ Inl)\ g\ h\ conv1$   
 (is ?lhs = ?rhs)  
 ⟨proof⟩

**lemma** *left-interface-parallel-converter*:

$left\text{-}interface\ (conv1 \mid_{\infty} conv2) = left\text{-}interface\ conv1 \mid_{\infty} left\text{-}interface\ conv2$   
 ⟨proof⟩

**lemma** *right-interface-parallel-converter*:

$right\text{-}interface\ (conv1 \mid_{\infty} conv2) = right\text{-}interface\ conv1 \mid_{\infty} right\text{-}interface\ conv2$   
 ⟨proof⟩

**lemma** *left-interface-converter-of-resource [simp]*:

$left\text{-}interface\ (converter\text{-}of\text{-}resource\ res) = converter\text{-}of\text{-}resource\ res$   
 ⟨proof⟩

**lemma** *right-interface-converter-of-resource [simp]*:

$right\text{-}interface\ (converter\text{-}of\text{-}resource\ res) = converter\text{-}of\text{-}resource\ res$   
 ⟨proof⟩

**lemma** *parallel-converter-swap*:  $map\text{-}converter\ swap\text{-}sum\ swap\text{-}sum\ id\ id\ (conv1 \mid_{\infty} conv2) = conv2 \mid_{\infty} conv1$

⟨proof⟩

**lemma** *eq-I-converter-map-converter'*:

assumes  $\mathcal{I}''$ ,  $map\text{-}\mathcal{I}\ f'\ g'\ \mathcal{I}' \vdash_C conv1 \sim conv2$   
 and  $f' \text{ 'outs-}\mathcal{I}\ \mathcal{I} \subseteq \text{outs-}\mathcal{I}\ \mathcal{I}''$

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

**lemma** *parallel-converter-eq- $\mathcal{I}$ -cong*:

$\llbracket \mathcal{I}1, \mathcal{I} \vdash_C \text{conv1} \sim \text{conv1}' ; \mathcal{I}2, \mathcal{I} \vdash_C \text{conv2} \sim \text{conv2}' \rrbracket$   
 $\implies \mathcal{I}1 \oplus_{\mathcal{I}} \mathcal{I}2, \mathcal{I} \vdash_C \text{parallel-converter } \text{conv1} \ \text{conv2} \sim \text{parallel-converter } \text{conv1}' \ \text{conv2}'$   
 $\langle \text{proof} \rangle$

**lemma**

*exec-gpv-parallel-oracle-right*:

$\text{exec-gpv} (\text{oracle1} \ddagger_O \text{oracle2}) (\text{right-gpv } \text{gpv}) \ s = \text{exec-gpv} (\dagger \text{oracle2}) \ \text{gpv} \ s$   
 $\langle \text{proof} \rangle$

**lemma**

*exec-gpv-parallel-oracle-left*:

$\text{exec-gpv} (\text{oracle1} \ddagger_O \text{oracle2}) (\text{left-gpv } \text{gpv}) \ s = \text{exec-gpv} (\text{oracle1} \dagger) \ \text{gpv} \ s$  (**is** ?L  
= ?R)  
 $\langle \text{proof} \rangle$

**end**

**theory** *Observe-Failure imports*

*More-CC*

**begin**

**declare**  $\llbracket \text{show-variants} \rrbracket$

**context** *fixes*  $\text{oracle} :: ('s, 'in, 'out) \ \text{oracle}'$  **begin**

**fun** *obsf-oracle*  $:: ('s \ \text{exception}, 'in, 'out \ \text{exception}) \ \text{oracle}'$  **where**

$\text{obsf-oracle} \ \text{Fault} \ x = \text{return-spmf} \ (\text{Fault}, \ \text{Fault})$   
 $| \ \text{obsf-oracle} \ (\text{OK} \ s) \ x = \text{TRY} \ \text{map-spmf} \ (\text{map-prod} \ \text{OK} \ \text{OK}) \ (\text{oracle} \ s \ x) \ \text{ELSE}$   
 $\text{return-spmf} \ (\text{Fault}, \ \text{Fault})$

**end**

**type-synonym**  $('a, 'b) \ \text{resource-obsf} = ('a, 'b \ \text{exception}) \ \text{resource}$

**translations**

$(\text{type}) \ ('a, 'b) \ \text{resource-obsf} \leq (\text{type}) \ ('a, 'b \ \text{exception}) \ \text{resource}$

**primcorec** *obsf-resource*  $:: ('in, 'out) \ \text{resource} \Rightarrow ('in, 'out) \ \text{resource-obsf}$  **where**

$\text{run-resource} \ (\text{obsf-resource} \ \text{res}) = (\lambda x.$   
 $\text{map-spmf} \ (\text{map-prod} \ \text{id} \ \text{obsf-resource})$   
 $\ (\text{map-spmf} \ (\text{map-prod} \ \text{id} \ (\lambda \text{resF}. \ \text{case} \ \text{resF} \ \text{of} \ \text{OK} \ \text{res}' \Rightarrow \ \text{res}' \ | \ \text{Fault} \Rightarrow$   
 $\ \text{fail-resource})))$   
 $\ (\text{TRY} \ \text{map-spmf} \ (\text{map-prod} \ \text{OK} \ \text{OK}) \ (\text{run-resource} \ \text{res} \ x) \ \text{ELSE} \ \text{return-spmf}$   
 $\ (\text{Fault}, \ \text{Fault})))$

**lemma** *obsf-resource-sel*:

*run-resource (obsf-resource res) x =*  
*map-spmf (map-prod id (λresF. 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 I (obsf-resource res)*  
*⟨proof⟩*

**lemma** *WT-obsf-resource* [*WT-intro, simp*]: *exception-I I ⊢ res obsf-resource res*  
*√ if 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 D = map-gpv' (λx. x = Some True) id option-of-exception*  
*(gpv-stop D)*

**lemma** *WT-obsf-distinguisher* [*WT-intro*]:

*exception-I I ⊢ g obsf-distinguisher A √ if [WT-intro]: I ⊢ g A √*  
*⟨proof⟩*

**lemma** *interaction-bounded-by-obsf-distinguisher* [*interaction-bound*]:

*interaction-bounded-by consider (obsf-distinguisher A) bound*  
**if** [*interaction-bound*]: *interaction-bounded-by consider A bound*

*<proof>*

**lemma** *plossless-obsf-distinguisher* [simp]:  
 *plossless-gpv* (*exception- $\mathcal{I}$*   $\mathcal{I}$ ) (*obsf-distinguisher*  $\mathcal{A}$ )  
 **if** *plossless-gpv*  $\mathcal{I}$   $\mathcal{A}$   $\mathcal{I} \vdash g$   $\mathcal{A}$   $\checkmark$   
 *<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*  $\Rightarrow$  (*'a*, *'b*, *'c*, *'d*) *converter-obsf*  
**where**

*run-converter* (*obsf-converter* *conv*) = ( $\lambda x$ .  
 *map-gpv* (*map-prod* *id* *obsf-converter*) *id*  
 (*map-gpv* ( $\lambda convF$ . *case* *convF* *of* *Fault*  $\Rightarrow$  (*Fault*, *fail-converter*) | *OK* (*a*, *conv'*)  
  $\Rightarrow$  (*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'* ( $\lambda y$ . *case* *y* *of* *None*  $\Rightarrow$  (*Fault*, *const-converter* *Fault*) | *Some*(*x*,  
 *conv'*)  $\Rightarrow$  (*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*  $\equiv$  *exec-gpv*  
 **and** *exec-gpv2*  $\equiv$  *exec-gpv*  
 **shows**  
 *exec-gpv1* *run-resource* (*map-gpv'* *id* *id* *option-of-exception* (*gpv-stop* *gpv*)) (*obsf-resource*  
 *res*)  $\uparrow$   $\{(Some\ x, y) \mid x\ y.\ True\} =$   
 *map- $\text{spm}f$*  (*map-prod* *Some* *obsf-resource*) (*exec-gpv2* *run-resource* *gpv* *res*)  
 (**is** *?lhs* = *?rhs*)  
 *<proof>*

**lemma** *obsf-attach*:

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

**lemma** *inline1-gpv-stop-obsf-converter*:  
**defines** *inline1a*  $\equiv$  *inline1*  
**and** *inline1b*  $\equiv$  *inline1*  
**shows**  $\text{bind-spmf} (\text{inline1a } \text{run-converter} (\text{map-gpv}' \text{id } \text{id } \text{option-of-exception}$   
*(gpv-stop gpv)*) (*obsf-converter* *conv*)  
 $(\lambda xy. \text{case } xy \text{ of } \text{Inl } (\text{None}, \text{conv}') \Rightarrow \text{return-pmf } \text{None} \mid \text{Inl } (\text{Some } x, \text{conv}')$   
 $\Rightarrow \text{return-spmf} (\text{Inl } (x, \text{conv}') \mid \text{Inr } y \Rightarrow \text{return-spmf} (\text{Inr } y)) =$   
 $\text{map-spmf} (\text{map-sum} (\text{apsnd } \text{obsf-converter}))$   
 $(\text{apsnd} (\text{map-prod} (\lambda \text{rvp } \text{input}. \text{case } \text{input} \text{ of } \text{Fault} \Rightarrow \text{Done } (\text{Fault}, \text{const-converter}$   
*Fault*)  $\mid$  *OK* *input'*  $\Rightarrow$   
 $\text{map-gpv}' (\lambda \text{res}. \text{case } \text{res} \text{ of } \text{None} \Rightarrow (\text{Fault}, \text{const-converter } \text{Fault}) \mid$   
 $\text{Some } (x, \text{conv}') \Rightarrow (\text{OK } x, \text{obsf-converter } \text{conv}')) \text{ id } \text{option-of-exception} (\text{try-gpv}$   
*(gpv-stop (rvp input')) (Done None))*)  
 $(\lambda \text{rvp } \text{input}. \text{case } \text{input} \text{ of } \text{Fault} \Rightarrow \text{Done } \text{None} \mid \text{OK } \text{input}' \Rightarrow \text{map-gpv}' \text{ id } \text{id}$   
*option-of-exception (gpv-stop (rvp input'))*))  
 $(\text{inline1b } \text{run-converter } \text{gpv } \text{conv})$   
**(is ?lhs = ?rhs)**  
 ⟨*proof*⟩

**lemma** *inline-gpv-stop-obsf-converter*:  
 $\text{bind-gpv} (\text{inline } \text{run-converter} (\text{map-gpv}' \text{id } \text{id } \text{option-of-exception} (\text{gpv-stop } \text{gpv}))$   
*(obsf-converter conv)*)  $(\lambda(x, \text{conv}'). \text{case } x \text{ of } \text{None} \Rightarrow \text{Fail} \mid \text{Some } x' \Rightarrow \text{Done } (x,$   
*conv')) =*  
 $\text{bind-gpv} (\text{map-gpv}' \text{id } \text{id } \text{option-of-exception} (\text{gpv-stop} (\text{inline } \text{run-converter}$   
*gpv conv)))*  $(\lambda x. \text{case } x \text{ of } \text{None} \Rightarrow \text{Fail} \mid \text{Some } (x', \text{conv}) \Rightarrow \text{Done } (\text{Some } x',$   
*obsf-converter conv)*)  
 ⟨*proof*⟩

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

$conv2) \sim comp-converter (obsf-converter conv1) (obsf-converter conv2)$   
 ⟨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]:  $A \vdash_C obsf-oracle callee1(Fault) \approx obsf-oracle$   
 $callee2(Fault)$   
 ⟨proof⟩

**lemma** *obsf-resource-eq-I-cong*:  $A \vdash_R obsf-resource res1 \sim obsf-resource res2$  **if**  $A$   
 $\vdash_R res1 \sim res2$   
 ⟨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**  $A \vdash_C callee1(s) \approx callee2(s')$   
**shows**  $A \vdash_C obsf-oracle callee1(OK s) \approx obsf-oracle callee2(OK s')$   
 ⟨proof⟩

**lemma** *trace-eq-obsf-resourceI*:  
**assumes**  $A \vdash_R res1 \approx res2$   
**shows**  $A \vdash_R obsf-resource res1 \approx obsf-resource res2$   
 ⟨proof⟩

**lemma** *spmf-run-obsf-oracle-obsf-distinguisher* [rule-format]:  
**defines**  $eg1 \equiv exec-gpv$  **and**  $eg2 \equiv exec-gpv$  **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*:  
 $spmf (connect-obsf (obsf-distinguisher A) (obsf-resource res)) True = spmf$   
 $(connect A res) True$   
 ⟨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*  
 $\langle$ *proof* $\rangle$

**end**

**theory** *Fold-Spmf*

**imports**

*More-CC*

**begin**

**primrec** (*transfer*)

*foldl-spmf* :: (*'b*  $\Rightarrow$  *'a*  $\Rightarrow$  *'b* *spmf*)  $\Rightarrow$  *'b* *spmf*  $\Rightarrow$  *'a* *list*  $\Rightarrow$  *'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* ( $\lambda a. f a x$ ))  
*xs*

**lemma** *foldl-spmf-return-pmf-None* [*simp*]:

*foldl-spmf* *f* (*return-pmf* *None*) *xs* = *return-pmf* *None*

$\langle$ *proof* $\rangle$

**lemma** *foldl-spmf-bind-spmf*: *foldl-spmf* *f* (*bind-spmf* *p* *g*) *xs* = *bind-spmf* *p* ( $\lambda a. f a$ ).  
*foldl-spmf* *f* (*g* *a*) *xs*

$\langle$ *proof* $\rangle$

**lemma** *bind-foldl-spmf-return*:

*bind-spmf* *p* ( $\lambda x. f a x$ ). *foldl-spmf* *f* (*return-spmf* *x*) *xs* = *foldl-spmf* *f* *p* *xs*

$\langle$ *proof* $\rangle$

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

$\langle$ *proof* $\rangle$

**lemma** *foldl-spmf-identity* [*simp*]: *foldl-spmf* ( $\lambda s x. f a x$ ). *return-spmf* *s*) *p* *xs* = *p*

$\langle$ *proof* $\rangle$

**lemma** *foldl-spmf-conv-foldl*:

*foldl-spmf* ( $\lambda s x. f a x$ ). *return-spmf* (*f* *s* *x*)) *p* *xs* = *map-spmf* ( $\lambda s. f a s$ ). *foldl* *f* *s* *xs*) *p*

$\langle$ *proof* $\rangle$

**lemma** *foldl-spmf-Cons'*:

*foldl-spmf* *f* (*return-spmf* *a*) (*x* # *xs*) = *bind-spmf* (*f* *a* *x*) ( $\lambda a'. f a x$ ). *foldl-spmf* *f*  
(*return-spmf* *a'*) *xs*)

$\langle$ *proof* $\rangle$

**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**  $foldl\text{-}spmf (\lambda a e. map\text{-}spmf h (g (f a) e)) acc es =$   
 $map\text{-}spmf h (foldl\text{-}spmf g (map\text{-}spmf f acc) es)$

(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**  $foldl\text{-}spmf (\lambda a e. map\text{-}spmf (f (p a)) (g (q a) e)) acc es =$   
 $bind\text{-}spmf acc (\lambda acc'. map\text{-}spmf (f (p acc')) (foldl\text{-}spmf g (return\text{-}spmf (q acc'))$   
 $es))$

(proof)

**lemma** *foldl-pair-constl:*  $foldl (\lambda s e. map\text{-}prod (\lambda-. c) (\lambda r. f r e) s) (c, sr) l =$

$Pair c (foldl (\lambda s e. f s e) sr l)$

(proof)

**lemma** *foldl-spmf-pair-left:*

$foldl\text{-}spmf (\lambda(l, r) e. map\text{-}spmf (\lambda l'. (l', r)) (f l e)) (return\text{-}spmf (l, r)) es =$   
 $map\text{-}spmf (\lambda l'. (l', r)) (foldl\text{-}spmf f (return\text{-}spmf l) es)$

(proof)

**lemma** *foldl-spmf-pair-left2:*

$foldl\text{-}spmf (\lambda(l, -) e. map\text{-}spmf (\lambda l'. (l', c')) (f l e)) (return\text{-}spmf (l, c)) es =$   
 $map\text{-}spmf (\lambda l'. (l', if es = [] then c else c')) (foldl\text{-}spmf f (return\text{-}spmf l) es)$

(proof)

**lemma** *foldl-pair-constr:*  $foldl (\lambda s e. map\text{-}prod (\lambda l. f l e) (\lambda-. c) s) (sl, c) l =$

$Pair (foldl (\lambda s e. f s e) sl l) c$

(proof)

**lemma** *foldl-spmf-pair-right:*

$foldl\text{-}spmf (\lambda(l, r) e. map\text{-}spmf (\lambda r'. (l, r')) (f r e)) (return\text{-}spmf (l, r)) es =$   
 $map\text{-}spmf (\lambda r'. (l, r')) (foldl\text{-}spmf f (return\text{-}spmf r) es)$

(proof)

**lemma** *foldl-spmf-pair-right2:*

$foldl\text{-}spmf (\lambda(-, r) e. map\text{-}spmf (\lambda r'. (c', r')) (f r e)) (return\text{-}spmf (c, r)) es =$   
 $map\text{-}spmf (\lambda r'. (if es = [] then c else c', r')) (foldl\text{-}spmf f (return\text{-}spmf r) es)$

(proof)

**lemma** *foldl-spmf-pair-right3*:

$foldl\text{-}spmf\ (\lambda(l, r)\ e.\ map\text{-}spmf\ (Pair\ (g\ e))\ (f\ r\ e))\ (return\text{-}spmf\ (l, r))\ es =$   
 $map\text{-}spmf\ (Pair\ (if\ es = []\ then\ l\ else\ g\ (last\ es)))\ (foldl\text{-}spmf\ f\ (return\text{-}spmf\ r)$   
 $es)$   
 $\langle proof \rangle$

**lemma** *foldl-pullout*:  $bind\text{-}spmf\ f\ (\lambda x.\ bind\text{-}spmf\ (foldl\text{-}spmf\ g\ init\ (events\ x))\ (\lambda y.$   
 $h\ x\ y)) =$

$bind\text{-}spmf\ (bind\text{-}spmf\ f\ (\lambda x.\ foldl\text{-}spmf\ (\lambda(l, r)\ e.\ map\text{-}spmf\ (Pair\ l)\ (g\ r\ e))$   
 $(map\text{-}spmf\ (Pair\ x)\ init)\ (events\ x)))$   
 $(\lambda(x, y).\ h\ x\ y)\ \mathbf{for}\ f\ g\ h\ init\ events$   
 $\langle proof \rangle$

**lemma** *bind-foldl-spmf-pair-append*:

$bind\text{-}spmf$   
 $(foldl\text{-}spmf\ (\lambda(x, y)\ e.\ map\text{-}spmf\ (apfst\ ((@)\ x))\ (f\ y\ e))\ (return\text{-}spmf\ (a\ @\ c,$   
 $b))\ es)$   
 $(\lambda(x, y).\ g\ x\ y) =$   
 $bind\text{-}spmf$   
 $(foldl\text{-}spmf\ (\lambda(x, y)\ e.\ map\text{-}spmf\ (apfst\ ((@)\ x))\ (f\ y\ e))\ (return\text{-}spmf\ (c, b))$   
 $es)$   
 $(\lambda(x, y).\ g\ (a\ @\ x)\ y)$   
 $\langle proof \rangle$

**lemma** *foldl-spmf-chain*:

$(foldl\text{-}spmf\ (\lambda(oevents, s\text{-}event)\ event.\ map\text{-}spmf\ (map\text{-}prod\ ((@)\ oevents)\ id)\ (fff\ s\text{-}event\ event))\ (return\text{-}spmf\ ([],\ s\text{-}event))\ ievents)$   
 $\ggg (\lambda(oevents, s\text{-}event').\ foldl\text{-}spmf\ ggg\ (return\text{-}spmf\ s\text{-}core)\ oevents$   
 $\ggg (\lambda s\text{-}core'.\ return\text{-}spmf\ (f\ s\text{-}core'\ s\text{-}event')))) =$   
 $foldl\text{-}spmf\ (\lambda(s\text{-}event, s\text{-}core)\ event.\ fff\ s\text{-}event\ event\ \ggg (\lambda(oevents, s\text{-}event').$   
 $map\text{-}spmf\ (Pair\ s\text{-}event')\ (foldl\text{-}spmf\ ggg\ (return\text{-}spmf\ s\text{-}core)\ oevents)))$   
 $(return\text{-}spmf\ (s\text{-}event, s\text{-}core))\ ievents$   
 $\ggg (\lambda(s\text{-}event', s\text{-}core').\ return\text{-}spmf\ (f\ s\text{-}core'\ s\text{-}event'))$   
 $\langle proof \rangle$

**primrec** *pauses* ::  $'a\ list \Rightarrow (unit, 'a, 'b)\ gpv\ \mathbf{where}$

$pauses\ [] = Done\ ()$   
 $| pauses\ (x\ \#\ xs) = Pause\ x\ (\lambda.\ pauses\ xs)$

**lemma** *WT-gpv-pauses [WT-intro]*:

$\mathcal{I} \vdash_g\ pauses\ xs\ \surd\ \mathbf{if}\ set\ xs \subseteq\ outs\text{-}\mathcal{I}\ \mathcal{I}$   
 $\langle proof \rangle$

**lemma** *exec-gpv-pauses*:

$exec\text{-}gpv\ callee\ (pauses\ xs)\ s =$   
 $map\text{-}spmf\ (Pair\ ())\ (foldl\text{-}spmf\ (map\text{-}fun\ id\ (map\text{-}fun\ id\ (map\text{-}spmf\ snd)))\ callee)$   
 $(return\text{-}spmf\ s)\ xs)$   
 $\langle proof \rangle$

```

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  $\lambda\mathcal{I} x. \mathcal{I} x \times UNIV$   $\langle proof \rangle$ 

lemma  $outs\text{-}\mathcal{I}\text{-}e\mathcal{I}[simp]$ :  $outs\text{-}\mathcal{I} (e\mathcal{I} \mathcal{I}) = outs\text{-}\mathcal{I} \mathcal{I}$ 
 $\langle proof \rangle$ 

lemma  $responses\text{-}\mathcal{I}\text{-}e\mathcal{I} [simp]$ :  $responses\text{-}\mathcal{I} (e\mathcal{I} \mathcal{I}) x = responses\text{-}\mathcal{I} \mathcal{I} x \times UNIV$ 
 $\langle proof \rangle$ 

lemma  $e\mathcal{I}\text{-}map\text{-}\mathcal{I}$ :  $e\mathcal{I} (map\text{-}\mathcal{I} f g \mathcal{I}) = map\text{-}\mathcal{I} f (apfst g) (e\mathcal{I} \mathcal{I})$ 
 $\langle proof \rangle$ 

lemma  $e\mathcal{I}\text{-}inverse [simp]$ :  $map\text{-}\mathcal{I} id fst (e\mathcal{I} \mathcal{I}) = \mathcal{I}$ 
 $\langle proof \rangle$ 
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  $\times$  'event list)
oracle'

```

**definition**

```

parallel-eoracle ::
  ('s1, 'e1, 'i1, 'o1) eoracle  $\Rightarrow$  ('s2, 'e2, 'i2, 'o2) eoracle  $\Rightarrow$ 
  ('s1  $\times$  's2, 'e1 + 'e2, 'i1 + 'i2, 'o1 + 'o2) eoracle

```

**where**

```

parallel-eoracle eoracle1 eoracle2 state  $\equiv$ 
  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 ::

```

$(\text{'s}, \text{'e1}, \text{'i1}, \text{'o1}) \text{ eoracle} \Rightarrow (\text{'s}, \text{'e2}, \text{'i2}, \text{'o2}) \text{ eoracle} \Rightarrow$   
 $(\text{'s}, \text{'e1} + \text{'e2}, \text{'i1} + \text{'i2}, \text{'o1} + \text{'o2}) \text{ eoracle}$

**where**

$\text{plus-eoracle eoracle1 eoracle2 state} \equiv$   
 $\text{comp}$   
 $(\text{map-spmf}$   
 $(\text{map-prod}$   
 $(\text{case-sum}$   
 $(\text{map-prod Inl (map Inl)})$   
 $(\text{map-prod Inr (map Inr)}))$   
 $\text{id})$   
 $(\text{plus-oracle eoracle1 eoracle2 state})$

**definition**

$\text{translate-eoracle} ::$

$(\text{'s-event}, \text{'e1}, \text{'e2 list}) \text{ oracle}' \Rightarrow (\text{'s-event} \times \text{'s}, \text{'e1}, \text{'i}, \text{'o}) \text{ eoracle} \Rightarrow$   
 $(\text{'s-event} \times \text{'s}, \text{'e2}, \text{'i}, \text{'o}) \text{ eoracle}$

**where**

$\text{translate-eoracle translator eoracle state inp} \equiv$

$\text{bind-spmf}$

$(\text{eoracle state inp})$

$(\lambda((\text{out}, \text{e-in}), \text{s}).$

$\text{let conc} = (\lambda(\text{es}, \text{st}) \text{e. map-spmf (map-prod ((@) \text{es}) id) (translator st e))$

$\text{in do} \{$

$(\text{e-out}, \text{s-event}) \leftarrow \text{foldl-spmf conc (return-spmf ([], \text{fst s})) e-in};$

$\text{return-spmf} ((\text{out}, \text{e-out}), \text{s-event}, \text{snd s})$

$\} )$

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

**type-synonym**

$(\text{'state}, \text{'event}) \text{ handler} = \text{'state} \Rightarrow \text{'event} \Rightarrow \text{'state} \text{ spmf}$

**fun**

$\text{parallel-handler} :: (\text{'s1}, \text{'e1}) \text{ handler} \Rightarrow (\text{'s2}, \text{'e2}) \text{ handler} \Rightarrow (\text{'s1} \times \text{'s2}, \text{'e1} +$   
 $\text{'e2}) \text{ handler}$

**where**

$\text{parallel-handler left - s (Inl e1)} = \text{map-spmf} (\lambda \text{s1}'. (\text{s1}', \text{snd s})) (\text{left (fst s) e1})$

$| \text{parallel-handler - right s (Inr e2)} = \text{map-spmf} (\lambda \text{s2}'. (\text{fst s}, \text{s2}')) (\text{right (snd s)}$   
 $\text{e2})$

**definition**

$\text{plus-handler} :: (\text{'s}, \text{'e1}) \text{ handler} \Rightarrow (\text{'s}, \text{'e2}) \text{ handler} \Rightarrow (\text{'s}, \text{'e1} + \text{'e2}) \text{ handler}$

**where**

$\text{plus-handler left right s} \equiv \text{case-sum (left s) (right s)}$

**lemma parallel-handler-left:**

$\text{map-fun id (map-fun Inl id) (parallel-handler left right)} =$

$(\lambda(\text{s-l}, \text{s-r}) \text{q. map-spmf} (\lambda \text{s-l}'. (\text{s-l}', \text{s-r})) (\text{left s-l q}))$

*<proof>*

**lemma** *parallel-handler-right*:

*map-fun id (map-fun Inr id) (parallel-handler left right) =*  
*(λ(s-l, s-r) q. map-spmf (λs-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*]:

$$\begin{aligned} & ((X \text{====>} S \text{====>} E \text{====>} \text{rel-spmf } S) \text{====>} \\ & (X \text{====>} S \text{====>} IA \text{====>} \text{rel-spmf } (\text{rel-prod } OA \ S)) \text{====>} \\ & (X \text{====>} S \text{====>} IU \text{====>} \text{rel-spmf } (\text{rel-prod } OU \ S)) \text{====>} \\ & X \text{====>} \text{rel-core}' \ S \ E \ IA \ IU \ OA \ OU) \text{ corec-core corec-core} \\ & \langle \text{proof} \rangle \end{aligned}$$

**primcorec** *map-core'* ::

$$\begin{aligned} & ('event' \Rightarrow 'event') \Rightarrow \\ & ('iadv-core' \Rightarrow 'iadv-core') \Rightarrow \\ & ('iusr-core' \Rightarrow 'iusr-core') \Rightarrow \\ & ('oadv-core' \Rightarrow 'oadv-core') \Rightarrow \\ & ('ousr-core' \Rightarrow 'ousr-core') \Rightarrow \\ & ('s-core, 'event, 'iadv-core, 'iusr-core, 'oadv-core, 'ousr-core) \text{ core} \Rightarrow \\ & ('s-core, 'event', 'iadv-core', 'iusr-core', 'oadv-core', 'ousr-core) \text{ core} \end{aligned}$$

**where**

$$\begin{aligned} & \text{cpoke } (\text{map-core}' \ e \ ia \ iu \ oa \ ou \ \text{core}) = (\text{id} \text{---->} \ e \ \text{---->} \ \text{id}) \ (\text{cpoke } \ \text{core}) \\ & | \ \text{cfunc-adv } (\text{map-core}' \ e \ ia \ iu \ oa \ ou \ \text{core}) = (\text{id} \text{---->} \ ia \ \text{---->} \ \text{map-spmf} \\ & \ (\text{map-prod } oa \ \text{id})) \ (\text{cfunc-adv } \ \text{core}) \\ & | \ \text{cfunc-usr } (\text{map-core}' \ e \ ia \ iu \ oa \ ou \ \text{core}) = (\text{id} \text{---->} \ iu \ \text{---->} \ \text{map-spmf} \\ & \ (\text{map-prod } ou \ \text{id})) \ (\text{cfunc-usr } \ \text{core}) \end{aligned}$$

**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*:

$$\begin{aligned} & \text{rel-core}' \ (=) \ (\text{BNF-Def.Grp } UNIV \ e)^{-1-1} \ (\text{BNF-Def.Grp } UNIV \ ia)^{-1-1} \ (\text{BNF-Def.Grp} \\ & \ UNIV \ iu)^{-1-1} \ (\text{BNF-Def.Grp } UNIV \ oa) \ (\text{BNF-Def.Grp } UNIV \ ou) \\ & = \ \text{BNF-Def.Grp } UNIV \ (\text{map-core}' \ e \ ia \ iu \ oa \ ou) \\ & \langle \text{proof} \rangle \end{aligned}$$

**end**

**inductive** *WT-core* :: ('iadv, 'oadv)  $\mathcal{I}$   $\Rightarrow$  ('iusr, 'ousr)  $\mathcal{I}$   $\Rightarrow$  ('s  $\Rightarrow$  bool)  $\Rightarrow$  ('s, 'event, 'iadv, 'iusr, 'oadv, 'ousr) *core*  $\Rightarrow$  bool

**for**  $\mathcal{I}$ -adv  $\mathcal{I}$ -usr *I* *core* **where**

*WT-core*  $\mathcal{I}$ -adv  $\mathcal{I}$ -usr *I* *core* **if**

$$\bigwedge s \ e \ s'. \llbracket s' \in \text{set-spmf } (\text{cpoke } \text{core } s \ e); I \ s \rrbracket \Longrightarrow I \ s'$$

$$\bigwedge s \ x \ y \ s'. \llbracket (y, s') \in \text{set-spmf } (\text{cfunc-adv } \text{core } s \ x); x \in \text{outs-}\mathcal{I} \ \mathcal{I}\text{-adv}; I \ s \rrbracket \Longrightarrow y \in \text{responses-}\mathcal{I} \ \mathcal{I}\text{-adv } x \wedge I \ s'$$

$$\bigwedge s \ x \ y \ s'. \llbracket (y, s') \in \text{set-spmf } (\text{cfunc-usr } \text{core } s \ x); x \in \text{outs-}\mathcal{I} \ \mathcal{I}\text{-usr}; I \ s \rrbracket \Longrightarrow y \in \text{responses-}\mathcal{I} \ \mathcal{I}\text{-usr } x \wedge I \ s'$$

**lemma** *WT-coreD*:

**assumes** *WT-core*  $\mathcal{I}$ -adv  $\mathcal{I}$ -usr *I* *core*

**shows** *WT-coreD-cpoke*:  $\bigwedge s \ e \ s'. \llbracket s' \in \text{set-spmf } (\text{cpoke } \text{core } s \ e); I \ s \rrbracket \Longrightarrow I \ s'$

**and** *WT-coreD-cfunc-adv*:  $\bigwedge s \ x \ y \ s'. \llbracket (y, s') \in \text{set-spmf } (\text{cfunc-adv } \text{core } s \ x);$

$x \in \text{outs-}\mathcal{I} \ \mathcal{I}\text{-adv}; I \ s \ ] \implies y \in \text{responses-}\mathcal{I} \ \mathcal{I}\text{-adv } x \wedge I \ s'$   
**and**  $WT\text{-coreD-cfund-usr}: \bigwedge s \ x \ y \ s'. \llbracket (y, s') \in \text{set-spmf } (cfunc\text{-usr } core \ s \ x);$   
 $x \in \text{outs-}\mathcal{I} \ \mathcal{I}\text{-usr}; I \ s \ ] \implies y \in \text{responses-}\mathcal{I} \ \mathcal{I}\text{-usr } x \wedge I \ s'$   
 $\langle \text{proof} \rangle$

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

**lemma**  $WT\text{-core-trivial}$ :  
**assumes**  $adv: \bigwedge s. \mathcal{I}\text{-adv} \vdash c \ cfunc\text{-adv } core \ s \ \checkmark$   
**and**  $usr: \bigwedge s. \mathcal{I}\text{-usr} \vdash c \ cfunc\text{-usr } core \ s \ \checkmark$   
**shows**  $WT\text{-core } \mathcal{I}\text{-adv } \mathcal{I}\text{-usr} \ (\lambda-. \text{True}) \ core$   
 $\langle \text{proof} \rangle$

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

**declare**  $\text{rest-scheme.sel-transfer}[\text{transfer-rule } del]$   
**declare**  $\text{rest-scheme.ctr-transfer}[\text{transfer-rule } del]$   
**declare**  $\text{rest-scheme.case-transfer}[\text{transfer-rule } del]$

**context**  
**includes**  $\text{lifting-syntax}$   
**begin**

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

**if**  
 $M$  *rinit* *rinit'* **and**  
 $(S \implies IA \implies \text{rel-spmf} (\text{rel-prod} (\text{rel-prod } OA \text{ (list-all2 } E)) S))$  *rfunc-adv*  
*rfunc-adv'* **and**  
 $(S \implies IU \implies \text{rel-spmf} (\text{rel-prod} (\text{rel-prod } OU \text{ (list-all2 } E)) S))$  *rfunc-usr*  
*rfunc-usr'*  
**for** *rinit* *rfunc-adv* *rfunc-usr*

**inductive-simps**

*rel-rest'-simps* [*simp*]:  
 $\text{rel-rest}' S E IA IU OA OU M (\text{Rest } \text{rinit}' \text{rfunc-adv}' \text{rfunc-usr}') (\text{Rest } \text{rinit}'' \text{rfunc-adv}'' \text{rfunc-usr}'')$

**lemma**

*rel-rest'-eq* [*relator-eq*]:  
 $\text{rel-rest}' (=) (=) (=) (=) (=) (=) (=) (=)$   
*<proof>*

**lemma**

*rel-rest'-mono* [*relator-mono*]:  
 $\text{rel-rest}' S E IA IU OA OU M \leq \text{rel-rest}' S E' IA' IU' OA' OU' M'$   
**if**  $E \leq E' IA' \leq IA IU' \leq IU OA \leq OA' OU \leq OU' M \leq M'$   
*<proof>*

**lemma** *rel-rest'-sel*:  $\text{rel-rest}' S E IA IU OA OU M \text{rest1 rest2}$

**if**  $M$  (*rinit* *rest1*) (*rinit* *rest2*)  
**and**  $(S \implies IA \implies \text{rel-spmf} (\text{rel-prod} (\text{rel-prod } OA \text{ (list-all2 } E)) S))$   
(*rfunc-adv* *rest1*) (*rfunc-adv* *rest2*)  
**and**  $(S \implies IU \implies \text{rel-spmf} (\text{rel-prod} (\text{rel-prod } OU \text{ (list-all2 } E)) S))$   
(*rfunc-usr* *rest1*) (*rfunc-usr* *rest2*)  
*<proof>*

**lemma** *rinit-parametric* [*transfer-rule*]:  $(\text{rel-rest}' S E IA IU OA OU M \implies M)$   
*rinit* *rinit*  
*<proof>*

**lemma** *rfunc-adv-parametric* [*transfer-rule*]:

$(\text{rel-rest}' S E IA IU OA OU M \implies S \implies IA \implies \text{rel-spmf} (\text{rel-prod} (\text{rel-prod } OA \text{ (list-all2 } E)) S))$  *rfunc-adv* *rfunc-adv*  
*<proof>*

**lemma** *rfunc-usr-parametric* [*transfer-rule*]:

$(\text{rel-rest}' S E IA IU OA OU M \implies S \implies IU \implies \text{rel-spmf} (\text{rel-prod} (\text{rel-prod } OU \text{ (list-all2 } E)) S))$  *rfunc-usr* *rfunc-usr*  
*<proof>*

**lemma** *Rest-parametric* [*transfer-rule*]:

$(M \implies (S \implies IA \implies \text{rel-spmf} (\text{rel-prod} (\text{rel-prod } OA \text{ (list-all2 } E)) S))$   
 $S))$

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

**lemma** *case-rest-scheme-parametric* [transfer-rule]:

(( $M ====>$   
 $(S ====> IA ====> \text{rel-spmf } (\text{rel-prod } (\text{rel-prod } OA \text{ (list-all2 } E)) S)) ====>$   
 $(S ====> IU ====> \text{rel-spmf } (\text{rel-prod } (\text{rel-prod } OU \text{ (list-all2 } E)) S)) ====>$   
 $X) ====>$   
 $\text{rel-rest}' S E IA IU OA OU M ====> X)$  *case-rest-scheme case-rest-scheme*  
 <proof>

**lemma** *corec-rest-scheme-parametric* [transfer-rule]:

(( $X ====> M) ====>$   
 $(X ====> S ====> IA ====> \text{rel-spmf } (\text{rel-prod } (\text{rel-prod } OA \text{ (list-all2 } E))$   
 $S)) ====>$   
 $(X ====> S ====> IU ====> \text{rel-spmf } (\text{rel-prod } (\text{rel-prod } OU \text{ (list-all2 } E))$   
 $S)) ====>$   
 $X ====> \text{rel-rest}' S E IA IU OA OU M)$  *corec-rest-scheme corec-rest-scheme*  
 <proof>

**primcorec** *map-rest'* ::

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

**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*:

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

*<proof>*

**end**

**type-synonym**

$(\text{'s-rest}, \text{'event}, \text{'iadv-rest}, \text{'iusr-rest}, \text{'oadv-rest}, \text{'ousr-rest}) \text{rest-wstate} =$   
 $(\text{'s-rest}, \text{'event}, \text{'iadv-rest}, \text{'iusr-rest}, \text{'oadv-rest}, \text{'ousr-rest}, \text{'s-rest}) \text{rest-scheme}$

**inductive**  $WT\text{-rest} :: (\text{'iadv}, \text{'oadv}) \mathcal{I} \Rightarrow (\text{'iusr}, \text{'ousr}) \mathcal{I} \Rightarrow (\text{'s} \Rightarrow \text{bool}) \Rightarrow (\text{'s},$   
 $\text{'event}, \text{'iadv}, \text{'iusr}, \text{'oadv}, \text{'ousr}) \text{rest-wstate} \Rightarrow \text{bool}$

**for**  $\mathcal{I}\text{-adv} \mathcal{I}\text{-usr} I \text{rest}$  **where**

$WT\text{-rest} \mathcal{I}\text{-adv} \mathcal{I}\text{-usr} I \text{rest}$  **if**

$\bigwedge s x y es s'. \llbracket ((y, es), s') \in \text{set-spmf} (\text{rfunc-adv rest } s x); x \in \text{outs-}\mathcal{I} \mathcal{I}\text{-adv}; I s$   
 $\rrbracket \Longrightarrow y \in \text{responses-}\mathcal{I} \mathcal{I}\text{-adv } x \wedge I s'$

$\bigwedge s x y es s'. \llbracket ((y, es), s') \in \text{set-spmf} (\text{rfunc-usr rest } s x); x \in \text{outs-}\mathcal{I} \mathcal{I}\text{-usr}; I s$   
 $\rrbracket \Longrightarrow y \in \text{responses-}\mathcal{I} \mathcal{I}\text{-usr } x \wedge I s'$

$I (\text{rinit rest})$

**lemma**  $WT\text{-restD}$ :

**assumes**  $WT\text{-rest} \mathcal{I}\text{-adv} \mathcal{I}\text{-usr} I \text{rest}$

**shows**  $WT\text{-restD-rfunc-adv}: \bigwedge s x y es s'. \llbracket ((y, es), s') \in \text{set-spmf} (\text{rfunc-adv}$   
 $\text{rest } s x); x \in \text{outs-}\mathcal{I} \mathcal{I}\text{-adv}; I s \rrbracket \Longrightarrow y \in \text{responses-}\mathcal{I} \mathcal{I}\text{-adv } x \wedge I s'$

**and**  $WT\text{-restD-rfunc-usr}: \bigwedge s x y es s'. \llbracket ((y, es), s') \in \text{set-spmf} (\text{rfunc-usr rest}$   
 $s x); x \in \text{outs-}\mathcal{I} \mathcal{I}\text{-usr}; I s \rrbracket \Longrightarrow y \in \text{responses-}\mathcal{I} \mathcal{I}\text{-usr } x \wedge I s'$

**and**  $WT\text{-restD-rinit}: I (\text{rinit rest})$

*<proof>*

**abbreviation**

$\text{fuse-cfunc} ::$

$(\text{'o} \Rightarrow \text{'x}) \Rightarrow (\text{'s-core}, \text{'i}, \text{'o}) \text{oracle}' \Rightarrow (\text{'s-core} \times \text{'s-rest}, \text{'i}, \text{'x}) \text{oracle}'$

**where**

$\text{fuse-cfunc redirect cfunc state inp} \equiv \text{do} \{$   
 $\text{let handle} = \text{map-prod redirect (prod.swap o Pair (snd state))};$   
 $(\text{os-cfunc} :: \text{'o} \times \text{'s-core}) \leftarrow \text{cfunc (fst state) inp};$   
 $\text{return-spmf (handle os-cfunc)}$   
 $\}$

**abbreviation**

$\text{fuse-rfunc} ::$

$(\text{'o} \Rightarrow \text{'x}) \Rightarrow (\text{'s-rest}, \text{'e}, \text{'i}, \text{'o}) \text{eoracle} \Rightarrow (\text{'s-core}, \text{'e}) \text{handler} \Rightarrow$   
 $(\text{'s-core} \times \text{'s-rest}, \text{'i}, \text{'x}) \text{oracle}'$

**where**

$\text{fuse-rfunc redirect rfunc notify state inp} \equiv$

$\text{bind-spmf}$   
 $(\text{rfunc (snd state) inp})$   
 $(\lambda((\text{o-rfunc}, \text{e-lst}), \text{s-rfunc}).$   
 $\text{bind-spmf}$   
 $(\text{foldl-spmf notify (return-spmf (fst state)) e-lst}$   
 $(\lambda\text{s-notify. return-spmf (redirect o-rfunc, s-notify, s-rfunc))))$

```

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  $\Rightarrow$ 
      ('s-core  $\times$  '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) (( $\mathcal{I}$ -adv-core  $\oplus_{\mathcal{I}}$ 
 $\mathcal{I}$ -adv-rest)  $\oplus_{\mathcal{I}}$  ( $\mathcal{I}$ -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  $\Rightarrow$ 
      (('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 ( $\mathcal{I}$ -adv-core  $\oplus_{\mathcal{I}}$   $\mathcal{I}$ -adv-rest)  $\oplus_{\mathcal{I}}$  ( $\mathcal{I}$ -usr-core  $\oplus_{\mathcal{I}}$   $\mathcal{I}$ -usr-rest)  $\vdash_{res}$  resource
    rest  $\checkmark$ 
    <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*:  $\text{cfunc-adv } (\text{attach-core } 1_I \text{ } \text{cnv } \text{core}) =$   
 $(\lambda(s\text{-cnv}, s\text{-core}) q. \text{map-spmf } (\lambda(\text{out}, s\text{-core}'). (\text{out}, s\text{-cnv}, s\text{-core}')) (\text{cfunc-adv}$   
 $\text{core } s\text{-core } q))$   
 $\langle \text{proof} \rangle$

**lemma**

*attach-core-id-oracle-usr*:  $\text{cfunc-usr } (\text{attach-core } \text{cnv } 1_I \text{ } \text{core}) =$   
 $(\lambda(s\text{-cnv}, s\text{-core}) q. \text{map-spmf } (\lambda(\text{out}, s\text{-core}'). (\text{out}, s\text{-cnv}, s\text{-core}')) (\text{cfunc-usr}$   
 $\text{core } s\text{-core } q))$   
 $\langle \text{proof} \rangle$

**context****fixes**

$\text{rest1} :: ('s\text{-rest1}, 'event1, 'iadv\text{-rest1}, 'iusr\text{-rest1}, 'oadv\text{-rest1}, 'ousr\text{-rest1}, 'more1)$   
 $\text{rest-scheme}$  **and**  
 $\text{rest2} :: ('s\text{-rest2}, 'event2, 'iadv\text{-rest2}, 'iusr\text{-rest2}, 'oadv\text{-rest2}, 'ousr\text{-rest2}, 'more2)$   
 $\text{rest-scheme}$

**begin****primcorec** *parallel-rest* ::

$( 's\text{-rest1} \times 's\text{-rest2}, 'event1 + 'event2, 'iadv\text{-rest1} + 'iadv\text{-rest2}, 'iusr\text{-rest1} +$   
 $'iusr\text{-rest2},$   
 $'oadv\text{-rest1} + 'oadv\text{-rest2}, 'ousr\text{-rest1} + 'ousr\text{-rest2}, 'more1 \times 'more2) \text{rest-scheme}$

**where**

$\text{rinit } \text{parallel-rest} = (\text{rinit } \text{rest1}, \text{rinit } \text{rest2})$   
 $| \text{rfunc-adv } \text{parallel-rest} = \text{parallel-eoracle } (\text{rfunc-adv } \text{rest1}) (\text{rfunc-adv } \text{rest2})$   
 $| \text{rfunc-usr } \text{parallel-rest} = \text{parallel-eoracle } (\text{rfunc-usr } \text{rest1}) (\text{rfunc-usr } \text{rest2})$

**end****lemma** *WT-parallel-rest* [*WT-intro*]:

$\text{WT-rest } (\mathcal{I}\text{-adv1} \oplus_{\mathcal{I}} \mathcal{I}\text{-adv2}) (\mathcal{I}\text{-usr1} \oplus_{\mathcal{I}} \mathcal{I}\text{-usr2}) (\text{pred-prod } I1 \ I2) (\text{parallel-rest}$   
 $\text{rest1 } \text{rest2})$   
**if**  $\text{WT-rest } \mathcal{I}\text{-adv1 } \mathcal{I}\text{-usr1 } I1 \ \text{rest1}$   
**and**  $\text{WT-rest } \mathcal{I}\text{-adv2 } \mathcal{I}\text{-usr2 } I2 \ \text{rest2}$   
 $\langle \text{proof} \rangle$

**context****fixes**

$\text{cnv-adv} :: 's\text{-adv} \Rightarrow 'iadv \Rightarrow ('oadv \times 's\text{-adv}, 'iadv\text{-rest}, 'oadv\text{-rest}) \text{gpv}$  **and**  
 $\text{cnv-usr} :: 's\text{-usr} \Rightarrow 'iusr \Rightarrow ('ousr \times 's\text{-usr}, 'iusr\text{-rest}, 'ousr\text{-rest}) \text{gpv}$  **and**  
 $\text{f-init} :: 'more \Rightarrow 'more'$  **and**  
 $\text{rest} :: ('s\text{-rest}, 'event, 'iadv\text{-rest}, 'iusr\text{-rest}, 'oadv\text{-rest}, 'ousr\text{-rest}, 'more) \text{rest-scheme}$   
**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 = λorc (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 = λorc (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 1I 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 1I 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 [] =
    ( $\lambda e.$  weight-spmf' (bind-spmf S ( $\lambda s.$  cpoke core s e)),
      $\lambda ia.$  bind-spmf S ( $\lambda s.$  map-spmf fst (cfunc-adv core s ia)),
      $\lambda iu.$  bind-spmf S ( $\lambda s.$  map-spmf fst (cfunc-usr core s iu)))
| trace-core' S (obs # tr) = (case obs of
  Inl e  $\Rightarrow$  trace-core' (mk-lossless (bind-spmf S ( $\lambda s.$  cpoke core s e))) tr
  | Inr (Inl (ia, oa))  $\Rightarrow$  trace-core' (cond-spmf-fst (bind-spmf S ( $\lambda s.$  cfunc-adv core s ia)) oa) tr
  | Inr (Inr (iu, ou))  $\Rightarrow$  trace-core' (cond-spmf-fst (bind-spmf S ( $\lambda s.$  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
 $\Rightarrow$  ('s-core', 'event, 'iadv-core, 'iusr-core, 'oadv-core, 'ousr-core) core
 $\Rightarrow$  'event set  $\Rightarrow$  'iadv-core set  $\Rightarrow$  'iusr-core set
 $\Rightarrow$  's-core spmf  $\Rightarrow$  's-core' spmf  $\Rightarrow$  bool where
  trace-core-eq core1 core2 E IA IU p q  $\longleftrightarrow$ 
  ( $\forall tr.$  set tr  $\subseteq$  E  $\langle + \rangle$  (IA  $\times$  UNIV)  $\langle + \rangle$  (IU  $\times$  UNIV)  $\longrightarrow$ 
   rel-prod (eq-onp ( $\lambda e.$  e  $\in$  E) =====> (=)) (rel-prod (eq-onp ( $\lambda ia.$  ia  $\in$  IA) =====>
  (=)) (eq-onp ( $\lambda iu.$  iu  $\in$  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  $\subseteq$  E  $\langle + \rangle$  (IA  $\times$  UNIV)  $\langle + \rangle$  (IU  $\times$  UNIV)
shows trace-core-eqD-cpoke:
  e  $\in$  E  $\Longrightarrow$  fst (trace-core' core1 p tr) e = fst (trace-core' core2 q tr) e
and trace-core-eqD-cfunc-adv:
  ia  $\in$  IA  $\Longrightarrow$  fst (snd (trace-core' core1 p tr)) ia = fst (snd (trace-core' core2

```

$q\ tr))\ ia$   
**and** *trace-core-eqD-cfunc-usr*:  
 $iu \in IU \implies \text{snd} (\text{snd} (\text{trace-core}'\ core1\ p\ tr))\ iu = \text{snd} (\text{snd} (\text{trace-core}'\ core2\ q\ tr))\ iu$   
 $\langle \text{proof} \rangle$

**lemma** *trace-core-eqI*:

**assumes**  $\bigwedge tr\ e. \llbracket \text{set } tr \subseteq E \langle + \rangle (IA \times UNIV) \langle + \rangle (IU \times UNIV); e \in E \rrbracket$   
 $\implies \text{fst} (\text{trace-core}'\ core1\ p\ tr)\ e = \text{fst} (\text{trace-core}'\ core2\ q\ tr)\ e$   
**and**  $\bigwedge tr\ ia. \llbracket \text{set } tr \subseteq E \langle + \rangle (IA \times UNIV) \langle + \rangle (IU \times UNIV); ia \in IA \rrbracket$   
 $\implies \text{fst} (\text{snd} (\text{trace-core}'\ core1\ p\ tr))\ ia = \text{fst} (\text{snd} (\text{trace-core}'\ core2\ q\ tr))\ ia$   
**and**  $\bigwedge tr\ iu. \llbracket \text{set } tr \subseteq E \langle + \rangle (IA \times UNIV) \langle + \rangle (IU \times UNIV); iu \in IU \rrbracket$   
 $\implies \text{snd} (\text{snd} (\text{trace-core}'\ core1\ p\ tr))\ iu = \text{snd} (\text{snd} (\text{trace-core}'\ core2\ q\ tr))\ iu$   
**shows** *trace-core-eq core1 core2 E IA IU p q*  
 $\langle \text{proof} \rangle$

**lemma** *trace-core-return-pmf-None [simp]*:

$\text{trace-core}'\ core (\text{return-pmf } None)\ tr = (\lambda-. 0, \lambda-. \text{return-pmf } None, \lambda-. \text{return-pmf } None)$   
 $\langle \text{proof} \rangle$

**lemma** *rel-core'-into-trace-core-eq*: *trace-core-eq core core' E IA IU p q*

**if** *rel-core'*  $S (\text{eq-onp } (\lambda e. e \in E)) (\text{eq-onp } (\lambda ia. ia \in IA)) (\text{eq-onp } (\lambda iu. iu \in IU))$   
 $(=) (=)\ core\ core'$   
 $\text{rel-spmf } S\ p\ q$   
 $\langle \text{proof} \rangle$

**lemma** *trace-core-eq-simI*:

**fixes**  $core1 :: ('s\text{-core}, 'event, 'iadv\text{-core}, 'iusr\text{-core}, 'oadv\text{-core}, 'ousr\text{-core})\ core$   
**and**  $core2 :: ('s\text{-core}', 'event', 'iadv\text{-core}', 'iusr\text{-core}', 'oadv\text{-core}', 'ousr\text{-core}')\ core$   
**and**  $S :: 's\text{-core}\ \text{spmf} \Rightarrow 's\text{-core}'\ \text{spmf} \Rightarrow \text{bool}$   
**assumes** *start*:  $S\ p\ q$   
**and** *step-cpoke*:  $\bigwedge p\ q\ e. \llbracket S\ p\ q; e \in E \rrbracket \implies$   
 $\text{weight-spmf} (\text{bind-spmf } p (\lambda s. \text{cpoke } core1\ s\ e)) = \text{weight-spmf} (\text{bind-spmf } q (\lambda s. \text{cpoke } core2\ s\ e))$   
**and** *sim-cpoke*:  $\bigwedge p\ q\ e. \llbracket S\ p\ q; e \in E \rrbracket \implies$   
 $S (\text{mk-lossless} (\text{bind-spmf } p (\lambda s. \text{cpoke } core1\ s\ e))) (\text{mk-lossless} (\text{bind-spmf } q (\lambda s. \text{cpoke } core2\ s\ e)))$   
**and** *step-cfunc-adv*:  $\bigwedge p\ q\ ia. \llbracket S\ p\ q; ia \in IA \rrbracket \implies$   
 $\text{bind-spmf } p (\lambda s1. \text{map-spmf } \text{fst} (\text{cfunc-adv } core1\ s1\ ia)) = \text{bind-spmf } q (\lambda s2. \text{map-spmf } \text{fst} (\text{cfunc-adv } core2\ s2\ ia))$   
**and** *sim-cfunc-adv*:  $\bigwedge p\ q\ ia\ s1\ s2\ s1'\ s2'\ oa. \llbracket S\ p\ q; ia \in IA;$   
 $s1 \in \text{set-spmf } p; s2 \in \text{set-spmf } q; (oa, s1') \in \text{set-spmf} (\text{cfunc-adv } core1\ s1\ ia);$   
 $(oa, s2') \in \text{set-spmf} (\text{cfunc-adv } core2\ s2\ ia) \rrbracket$   
 $\implies S (\text{cond-spmf-fst} (\text{bind-spmf } p (\lambda s1. \text{cfunc-adv } core1\ s1\ ia))\ oa) (\text{cond-spmf-fst} (\text{bind-spmf } q (\lambda s2. \text{cfunc-adv } core2\ s2\ ia))\ oa)$   
**and** *step-cfunc-usr*:  $\bigwedge p\ q\ iu. \llbracket S\ p\ q; iu \in IU \rrbracket \implies$   
 $\text{bind-spmf } p (\lambda s1. \text{map-spmf } \text{fst} (\text{cfunc-usr } core1\ s1\ iu)) = \text{bind-spmf } q (\lambda s2. \text{map-spmf } \text{fst} (\text{cfunc-usr } core2\ s2\ iu))$

**and** *sim-cfunc-usr*:  $\bigwedge p q iu s1 s2 s1' s2' ou. \llbracket S p q; iu \in IU;$   
 $s1 \in \text{set-spmf } p; s2 \in \text{set-spmf } q; (ou, s1') \in \text{set-spmf } (\text{cfunc-usr core1 } s1 iu);$   
 $(ou, s2') \in \text{set-spmf } (\text{cfunc-usr core2 } s2 iu) \rrbracket$   
 $\implies S (\text{cond-spmf-fst } (\text{bind-spmf } p (\lambda s1. \text{cfunc-usr core1 } s1 iu)) ou) (\text{cond-spmf-fst}$   
 $(\text{bind-spmf } q (\lambda s2. \text{cfunc-usr core2 } s2 iu)) ou)$   
**shows** *trace-core-eq core1 core2 E IA IU p q*  
 $\langle \text{proof} \rangle$

**context**

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

**fun** *trace-core-aux*

$:: 's\text{-core } \text{spm}f \Rightarrow ('event + 'iadv\text{-core} \times 'oadv\text{-core} + 'iusr\text{-core} \times 'ousr\text{-core}) \text{list}$   
 $\Rightarrow 's\text{-core } \text{spm}f$  **where**  
 $\text{trace-core-aux } p \llbracket = p$   
 $\mid \text{trace-core-aux } p (\text{Inl } e \# tr) = \text{trace-core-aux } (\text{mk-lossless } (\text{bind-spmf } p (\lambda s. \text{cpoke}$   
 $\text{core } s e))) tr$   
 $\mid \text{trace-core-aux } p (\text{Inr } (\text{Inl } (ia, oa)) \# tr) = \text{trace-core-aux } (\text{cond-spmf-fst } (\text{bind-spmf}$   
 $p (\lambda s. \text{cfunc-adv core } s ia)) oa) tr$   
 $\mid \text{trace-core-aux } p (\text{Inr } (\text{Inr } (iu, ou)) \# tr) = \text{trace-core-aux } (\text{cond-spmf-fst } (\text{bind-spmf}$   
 $p (\lambda s. \text{cfunc-usr core } s iu)) ou) tr$

**end**

**lemma** *trace-core-conv-trace-core-aux*:

$\text{trace-core}' \text{ core } p tr =$   
 $(\lambda e. \text{weight-spmf } (\text{bind-spmf } (\text{trace-core-aux core } p tr) (\lambda s. \text{cpoke core } s e)),$   
 $\lambda ia. \text{bind-spmf } (\text{trace-core-aux core } p tr) (\lambda s. \text{map-spmf fst } (\text{cfunc-adv core } s$   
 $ia))),$   
 $\lambda iu. \text{bind-spmf } (\text{trace-core-aux core } p tr) (\lambda s. \text{map-spmf fst } (\text{cfunc-usr core } s$   
 $iu)))$   
 $\langle \text{proof} \rangle$

**lemma** *trace-core-aux-append*:

$\text{trace-core-aux core } p (tr @ tr') = \text{trace-core-aux core } (\text{trace-core-aux core } p tr)$   
 $tr'$   
 $\langle \text{proof} \rangle$

**inductive** *trace-core-closure*

$:: ('s\text{-core}, 'event, 'iadv\text{-core}, 'iusr\text{-core}, 'oadv\text{-core}, 'ousr\text{-core}) \text{core}$   
 $\Rightarrow ('s\text{-core}', 'event, 'iadv\text{-core}, 'iusr\text{-core}, 'oadv\text{-core}, 'ousr\text{-core}) \text{core}$   
 $\Rightarrow 'event \text{ set} \Rightarrow 'iadv\text{-core } \text{set} \Rightarrow 'iusr\text{-core } \text{set}$   
 $\Rightarrow 's\text{-core } \text{spm}f \Rightarrow 's\text{-core}' \text{spm}f \Rightarrow 's\text{-core } \text{spm}f \Rightarrow 's\text{-core}' \text{spm}f \Rightarrow \text{bool}$   
**for** *core1 core2 E IA IU p q where*  
 $\text{trace-core-closure core1 core2 E IA IU p q } (\text{trace-core-aux core1 } p tr) (\text{trace-core-aux}$   
 $\text{core2 } q tr)$   
**if**  $\text{set } tr \subseteq E \langle + \rangle IA \times UNIV \langle + \rangle IU \times UNIV$

**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*:  
 $e \in E \implies \text{weight-spmf } (\text{bind-spmf } p' (\lambda s. \text{cpoke core1 } s \ e)) = \text{weight-spmf } (\text{bind-spmf } q' (\lambda s. \text{cpoke core2 } s \ e))$   
 (is *PROP ?thesis1*)  
**and** *trace-core-closure-step-cfunc-adv*:  
 $ia \in IA \implies \text{bind-spmf } p' (\lambda s1. \text{map-spmf fst } (\text{cfunc-adv core1 } s1 \ ia)) = \text{bind-spmf } q' (\lambda s2. \text{map-spmf fst } (\text{cfunc-adv core2 } s2 \ ia))$   
 (is *PROP ?thesis2*)  
**and** *trace-core-closure-step-cfunc-usr*:  
 $iu \in IU \implies \text{bind-spmf } p' (\lambda s1. \text{map-spmf fst } (\text{cfunc-usr core1 } s1 \ iu)) = \text{bind-spmf } q' (\lambda s2. \text{map-spmf fst } (\text{cfunc-usr core2 } s2 \ iu))$   
 (is *PROP ?thesis3*)  
 ⟨*proof*⟩

**lemma** *trace-core-closure-sim*:

**fixes** *core1 core2 E IA IU p q*  
**defines**  $S \equiv \text{trace-core-closure core1 core2 E IA IU p q}$   
**assumes**  $S \ p' \ q'$   
**shows** *trace-core-closure-sim-cpoke*:  
 $e \in E \implies S \ (\text{mk-lossless } (\text{bind-spmf } p' (\lambda s. \text{cpoke core1 } s \ e))) \ (\text{mk-lossless } (\text{bind-spmf } q' (\lambda s. \text{cpoke core2 } s \ e)))$   
 (is *PROP ?thesis1*)  
**and** *trace-core-closure-sim-cfunc-adv*:  $ia \in IA$   
 $\implies S \ (\text{cond-spmf-fst } (\text{bind-spmf } p' (\lambda s1. \text{cfunc-adv core1 } s1 \ ia)) \ oa) \ (\text{cond-spmf-fst } (\text{bind-spmf } q' (\lambda s2. \text{cfunc-adv core2 } s2 \ ia)) \ oa)$   
 (is *PROP ?thesis2*)  
**and** *trace-core-closure-sim-cfunc-usr*:  $iu \in IU$   
 $\implies S \ (\text{cond-spmf-fst } (\text{bind-spmf } p' (\lambda s1. \text{cfunc-usr core1 } s1 \ iu)) \ ou) \ (\text{cond-spmf-fst } (\text{bind-spmf } q' (\lambda s2. \text{cfunc-usr core2 } s2 \ iu)) \ ou)$   
 (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**  $\bigwedge p \ q \ e. \llbracket S \ p \ q; e \in E \rrbracket \implies$   
 $\text{weight-spmf } (\text{bind-spmf } p (\lambda s. \text{cpoke core1 } s \ e)) = \text{weight-spmf } (\text{bind-spmf } q (\lambda s. \text{cpoke core2 } s \ e))$   
**and**  $\bigwedge p \ q \ e. \llbracket S \ p \ q; e \in E \rrbracket \implies$   
 $S \ (\text{mk-lossless } (\text{bind-spmf } p (\lambda s. \text{cpoke core1 } s \ e))) \ (\text{mk-lossless } (\text{bind-spmf } q (\lambda s. \text{cpoke core2 } s \ e)))$   
**and**  $\bigwedge p \ q \ ia. \llbracket S \ p \ q; ia \in IA \rrbracket \implies$

```

    bind-spmf p (λs1. map-spmf fst (cfunc-adv core1 s1 ia)) = bind-spmf q (λs2.
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 (λs1. cfunc-adv core1 s1 ia)) oa) (cond-spmf-fst
(bind-spmf q (λs2. cfunc-adv core2 s2 ia)) oa)
  and ∧ p q iu. [ S p q; iu ∈ IU ] ⇒
    bind-spmf p (λs1. map-spmf fst (cfunc-usr core1 s1 iu)) = bind-spmf q (λs2.
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 (λs1. cfunc-usr core1 s1 iu)) ou) (cond-spmf-fst
(bind-spmf q (λs2. 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 [] =
    (λia. bind-spmf S (λs. map-spmf fst (rfunc-adv rest s ia)),
    λiu. bind-spmf S (λs. 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 (λs. rfunc-adv rest s ia))
oa) tr
  | Inr (iu, ou) ⇒ trace-rest' (cond-spmf-fst (bind-spmf S (λs. 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*

$:: ('s\text{-rest}, 'event, 'iadv\text{-rest}, 'iusr\text{-rest}, 'oadv\text{-rest}, 'ousr\text{-rest}, 'more1) \text{ rest-scheme}$   
 $\Rightarrow ('s\text{-rest}', 'event, 'iadv\text{-rest}, 'iusr\text{-rest}, 'oadv\text{-rest}, 'ousr\text{-rest}, 'more2) \text{ rest-scheme}$   
 $\Rightarrow 'iadv\text{-rest set} \Rightarrow 'iusr\text{-rest set}$   
 $\Rightarrow 's\text{-rest spmf} \Rightarrow 's\text{-rest}' \text{ spmf} \Rightarrow \text{bool}$  **where**  
 $\text{trace-rest-eq rest1 rest2 IA IU p q} \iff$   
 $(\forall tr. \text{ set } tr \subseteq (IA \times UNIV) \lt+\gt (IU \times UNIV) \longrightarrow$   
 $\text{rel-prod (eq-onp } (\lambda ia. ia \in IA) \implies (=)) \text{ (eq-onp } (\lambda iu. iu \in IU) \implies (=))$   
 $\text{(trace-rest' rest1 p tr) (trace-rest' rest2 q tr)})$

**end**

**lemma** *trace-rest-eqD*:

**assumes** *trace-rest-eq rest1 rest2 IA IU p q*  
**and**  $\text{set } tr \subseteq (IA \times UNIV) \lt+\gt (IU \times UNIV)$   
**shows** *trace-rest-eqD-rfunc-adv*:  
 $ia \in IA \implies \text{fst (trace-rest' rest1 p tr) } ia = \text{fst (trace-rest' rest2 q tr) } ia$   
**and** *trace-rest-eqD-rfunc-usr*:  
 $iu \in IU \implies \text{snd (trace-rest' rest1 p tr) } iu = \text{snd (trace-rest' rest2 q tr) } iu$   
*<proof>*

**lemma** *trace-rest-eqI*:

**assumes**  $\bigwedge tr ia. \llbracket \text{set } tr \subseteq (IA \times UNIV) \lt+\gt (IU \times UNIV); ia \in IA \rrbracket$   
 $\implies \text{fst (trace-rest' rest1 p tr) } ia = \text{fst (trace-rest' rest2 q tr) } ia$   
**and**  $\bigwedge tr iu. \llbracket \text{set } tr \subseteq (IA \times UNIV) \lt+\gt (IU \times UNIV); iu \in IU \rrbracket$   
 $\implies \text{snd (trace-rest' rest1 p tr) } iu = \text{snd (trace-rest' rest2 q tr) } iu$   
**shows** *trace-rest-eq rest1 rest2 IA IU p q*  
*<proof>*

**lemma** *trace-rest-return-pmf-None [simp]*:

$\text{trace-rest' rest (return-pmf None) tr} = (\lambda-. \text{return-pmf None}, \lambda-. \text{return-pmf None})$   
*<proof>*

**lemma** *rel-rest'-into-trace-rest-eq*: *trace-rest-eq rest rest' IA IU p q*

**if**  $\text{rel-rest' } S (=) \text{ (eq-onp } (\lambda ia. ia \in IA)) \text{ (eq-onp } (\lambda iu. iu \in IU)) (=) (=) M \text{ rest}$   
 $\text{rest'}$   
 $\text{rel-spmf } S p q$   
*<proof>*

**lemma** *trace-rest-eq-simI*:

**fixes**  $\text{rest1} :: ('s\text{-rest}, 'event, 'iadv\text{-rest}, 'iusr\text{-rest}, 'oadv\text{-rest}, 'ousr\text{-rest}, 'more)$   
*rest-scheme*  
**and**  $\text{rest2} :: ('s\text{-rest}', 'event, 'iadv\text{-rest}, 'iusr\text{-rest}, 'oadv\text{-rest}, 'ousr\text{-rest}, 'more)$   
*rest-scheme*  
**and**  $S :: 's\text{-rest spmf} \Rightarrow 's\text{-rest}' \text{ spmf} \Rightarrow \text{bool}$   
**assumes** *start*:  $S p q$

**and** *step-rfunc-adv*:  $\bigwedge p q ia. \llbracket S p q; ia \in IA \rrbracket \implies$   
 $bind\text{-}spmf\ p\ (\lambda s1. map\text{-}spmf\ fst\ (rfunc\text{-}adv\ rest1\ s1\ ia)) = bind\text{-}spmf\ q\ (\lambda s2.$   
 $map\text{-}spmf\ fst\ (rfunc\text{-}adv\ rest2\ s2\ ia))$   
**and** *sim-rfunc-adv*:  $\bigwedge p q ia s1 s2 s1' s2' oa. \llbracket S p q; ia \in IA;$   
 $s1 \in set\text{-}spmf\ p; s2 \in set\text{-}spmf\ q; (oa, s1') \in set\text{-}spmf\ (rfunc\text{-}adv\ rest1\ s1\ ia);$   
 $(oa, s2') \in set\text{-}spmf\ (rfunc\text{-}adv\ rest2\ s2\ ia) \rrbracket$   
 $\implies S\ (cond\text{-}spmf\text{-}fst\ (bind\text{-}spmf\ p\ (\lambda s1. rfunc\text{-}adv\ rest1\ s1\ ia))\ oa)\ (cond\text{-}spmf\text{-}fst$   
 $(bind\text{-}spmf\ q\ (\lambda s2. rfunc\text{-}adv\ rest2\ s2\ ia))\ oa)$   
**and** *step-rfunc-usr*:  $\bigwedge p q iu. \llbracket S p q; iu \in IU \rrbracket \implies$   
 $bind\text{-}spmf\ p\ (\lambda s1. map\text{-}spmf\ fst\ (rfunc\text{-}usr\ rest1\ s1\ iu)) = bind\text{-}spmf\ q\ (\lambda s2.$   
 $map\text{-}spmf\ fst\ (rfunc\text{-}usr\ rest2\ s2\ iu))$   
**and** *sim-rfunc-usr*:  $\bigwedge p q iu s1 s2 s1' s2' ou. \llbracket S p q; iu \in IU;$   
 $s1 \in set\text{-}spmf\ p; s2 \in set\text{-}spmf\ q; (ou, s1') \in set\text{-}spmf\ (rfunc\text{-}usr\ rest1\ s1\ iu);$   
 $(ou, s2') \in set\text{-}spmf\ (rfunc\text{-}usr\ rest2\ s2\ iu) \rrbracket$   
 $\implies S\ (cond\text{-}spmf\text{-}fst\ (bind\text{-}spmf\ p\ (\lambda s1. rfunc\text{-}usr\ rest1\ s1\ iu))\ ou)\ (cond\text{-}spmf\text{-}fst$   
 $(bind\text{-}spmf\ q\ (\lambda s2. rfunc\text{-}usr\ rest2\ s2\ iu))\ ou)$   
**shows** *trace-rest-eq rest1 rest2 IA IU p q*  
 $\langle proof \rangle$

**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  $\Rightarrow$  ('iadv-rest  $\times$  'oadv-rest  $\times$  'event list + 'iusr-rest  $\times$  'ousr-rest  
 $\times$  'event list) list  $\Rightarrow$  's-rest spmf **where**  
 $trace\text{-}rest\text{-}aux\ p\ [] = p$   
 $| trace\text{-}rest\text{-}aux\ p\ (Inl\ (ia, oaes) \# tr) = trace\text{-}rest\text{-}aux\ (cond\text{-}spmf\text{-}fst\ (bind\text{-}spmf$   
 $p\ (\lambda s. rfunc\text{-}adv\ rest\ s\ ia))\ oaes)\ tr$   
 $| trace\text{-}rest\text{-}aux\ p\ (Inr\ (iu, oues) \# tr) = trace\text{-}rest\text{-}aux\ (cond\text{-}spmf\text{-}fst\ (bind\text{-}spmf$   
 $p\ (\lambda s. rfunc\text{-}usr\ rest\ s\ iu))\ oues)\ tr$

**end**

**lemma** *trace-rest-conv-trace-rest-aux*:

$trace\text{-}rest'\ rest\ p\ tr =$   
 $(\lambda ia. bind\text{-}spmf\ (trace\text{-}rest\text{-}aux\ rest\ p\ tr)\ (\lambda s. map\text{-}spmf\ fst\ (rfunc\text{-}adv\ rest\ s\ ia)),$   
 $\lambda iu. bind\text{-}spmf\ (trace\text{-}rest\text{-}aux\ rest\ p\ tr)\ (\lambda s. map\text{-}spmf\ fst\ (rfunc\text{-}usr\ rest\ s\ iu)))$   
 $\langle proof \rangle$

**lemma** *trace-rest-aux-append*:

$trace\text{-}rest\text{-}aux\ rest\ p\ (tr\ @\ tr') = trace\text{-}rest\text{-}aux\ rest\ (trace\text{-}rest\text{-}aux\ rest\ p\ tr)\ tr'$   
 $\langle proof \rangle$

**inductive** *trace-rest-closure*

:: ('s-rest, 'event, 'iadv-rest, 'iusr-rest, 'oadv-rest, 'ousr-rest, 'more) rest-scheme  
 $\Rightarrow$  ('s-rest', 'event', 'iadv-rest, 'iusr-rest, 'oadv-rest, 'ousr-rest, 'more') rest-scheme  
 $\Rightarrow$  'iadv-rest set  $\Rightarrow$  'iusr-rest set

$\Rightarrow 's\text{-rest\ spmf} \Rightarrow 's\text{-rest}'\ \text{spm}f \Rightarrow 's\text{-rest\ spmf} \Rightarrow 's\text{-rest}'\ \text{spm}f \Rightarrow \text{bool}$   
**for**  $\text{rest1 rest2 IA IU p q}$  **where**  
 $\text{trace-rest-closure rest1 rest2 IA IU p q (trace-rest-aux rest1 p tr) (trace-rest-aux rest2 q tr)}$   
**if**  $\text{set tr} \subseteq \text{IA} \times \text{UNIV} \langle + \rangle \text{IU} \times \text{UNIV}$

**lemma** *trace-rest-closure-start*:  $\text{trace-rest-closure rest1 rest2 IA IU p q p q}$   
 $\langle \text{proof} \rangle$

**lemma** *trace-rest-closure-step*:

**assumes**  $\text{trace-rest-eq rest1 rest2 IA IU p q}$   
**and**  $\text{trace-rest-closure rest1 rest2 IA IU p q p' q'}$   
**shows** *trace-rest-closure-step-rfunc-adv*:  
 $ia \in \text{IA} \implies \text{bind-spmf } p' (\lambda s1. \text{map-spmf fst (rfunc-adv rest1 s1 ia)}) = \text{bind-spmf } q' (\lambda s2. \text{map-spmf fst (rfunc-adv rest2 s2 ia)})$   
**(is PROP ?thesis1)**  
**and** *trace-rest-closure-step-rfunc-usr*:  
 $iu \in \text{IU} \implies \text{bind-spmf } p' (\lambda s1. \text{map-spmf fst (rfunc-usr rest1 s1 iu)}) = \text{bind-spmf } q' (\lambda s2. \text{map-spmf fst (rfunc-usr rest2 s2 iu)})$   
**(is PROP ?thesis2)**  
 $\langle \text{proof} \rangle$

**lemma** *trace-rest-closure-sim*:

**fixes**  $\text{rest1 rest2 IA IU p q}$   
**defines**  $S \equiv \text{trace-rest-closure rest1 rest2 IA IU p q}$   
**assumes**  $S p' q'$   
**shows** *trace-rest-closure-sim-rfunc-adv*:  $ia \in \text{IA}$   
 $\implies S (\text{cond-spmf-fst (bind-spmf } p' (\lambda s1. \text{rfunc-adv rest1 s1 ia})) oa)$   
 $(\text{cond-spmf-fst (bind-spmf } q' (\lambda s2. \text{rfunc-adv rest2 s2 ia})) oa)$   
**(is PROP ?thesis1)**  
**and** *trace-rest-closure-sim-rfunc-usr*:  $iu \in \text{IU}$   
 $\implies S (\text{cond-spmf-fst (bind-spmf } p' (\lambda s1. \text{rfunc-usr rest1 s1 iu})) ou)$   
 $(\text{cond-spmf-fst (bind-spmf } q' (\lambda s2. \text{rfunc-usr rest2 s2 iu})) ou)$   
**(is PROP ?thesis2)**  
 $\langle \text{proof} \rangle$

**proposition** *trace-rest-eq-complete*:

**assumes**  $\text{trace-rest-eq rest1 rest2 IA IU p q}$   
**obtains**  $S$   
**where**  $S p q$   
**and**  $\bigwedge p q ia. \llbracket S p q; ia \in \text{IA} \rrbracket \implies$   
 $\text{bind-spmf } p (\lambda s1. \text{map-spmf fst (rfunc-adv rest1 s1 ia)}) = \text{bind-spmf } q (\lambda s2. \text{map-spmf fst (rfunc-adv rest2 s2 ia)})$   
**and**  $\bigwedge p q ia oa. \llbracket S p q; ia \in \text{IA} \rrbracket$   
 $\implies S (\text{cond-spmf-fst (bind-spmf } p (\lambda s1. \text{rfunc-adv rest1 s1 ia})) oa) (\text{cond-spmf-fst (bind-spmf } q (\lambda s2. \text{rfunc-adv rest2 s2 ia})) oa)$   
**and**  $\bigwedge p q iu. \llbracket S p q; iu \in \text{IU} \rrbracket \implies$   
 $\text{bind-spmf } p (\lambda s1. \text{map-spmf fst (rfunc-usr rest1 s1 iu)}) = \text{bind-spmf } q (\lambda s2. \text{map-spmf fst (rfunc-usr rest2 s2 iu)})$

**and**  $\bigwedge p q iu ou. \llbracket S p q; iu \in IU \rrbracket$   
 $\implies S (\text{cond-spmf-fst } (\text{bind-spmf } p (\lambda s1. \text{rfunc-usr } \text{rest1 } s1 iu))) ou) (\text{cond-spmf-fst } (\text{bind-spmf } q (\lambda s2. \text{rfunc-usr } \text{rest2 } s2 iu))) ou)$   
 <proof>

**definition** *callee-of-core*

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

**lemma** *callee-of-core-simps* [simp]:

$\text{callee-of-core } \text{core } s (\text{Inl } e) = \text{map-spmf } (\text{Pair } (\text{Inl } ())) (\text{cpoke } \text{core } s e)$   
 $\text{callee-of-core } \text{core } s (\text{Inr } (\text{Inl } iadv\text{-core})) = \text{map-spmf } (\text{apfst } (\text{Inr} \circ \text{Inl})) (\text{cfunc-adv } \text{core } s iadv\text{-core})$   
 $\text{callee-of-core } \text{core } s (\text{Inr } (\text{Inr } iusr\text{-core})) = \text{map-spmf } (\text{apfst } (\text{Inr} \circ \text{Inr})) (\text{cfunc-usr } \text{core } s iusr\text{-core})$   
 <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)  $\vdash_c$  *callee-of-core* *core* *s*  $\checkmark$   
 <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*

$:: ('s\text{-rest}, 'event, 'iadv\text{-rest}, 'iusr\text{-rest}, 'oadv\text{-rest}, 'ousr\text{-rest}, 'more) \text{ rest-scheme}$   
 $\implies ('s\text{-rest}, 'iadv\text{-rest} + 'iusr\text{-rest}, 'oadv\text{-rest} \times 'event \text{ list} + 'ousr\text{-rest} \times 'event \text{ list})$  *oracle'* **where**  
 $\text{callee-of-rest } \text{rest} = \text{rfunc-adv } \text{rest} \oplus_O \text{rfunc-usr } \text{rest}$

**lemma** *callee-of-rest-simps* [simp]:

$\text{callee-of-rest } \text{rest } s (\text{Inl } iadv\text{-rest}) = \text{map-spmf } (\text{apfst } \text{Inl}) (\text{rfunc-adv } \text{rest } s iadv\text{-rest})$   
 $\text{callee-of-rest } \text{rest } s (\text{Inr } iusr\text{-rest}) = \text{map-spmf } (\text{apfst } \text{Inr}) (\text{rfunc-usr } \text{rest } s iusr\text{-rest})$   
 <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  $\vdash_c$  *callee-of-rest* *rest* *s*  $\checkmark$

*<proof>*

**fun** *fuse-callee*

$:: ('iadv-core + 'iadv-rest) + ('iusr-core + 'iusr-rest) \Rightarrow$   
 $(('oadv-core + 'oadv-rest) + ('ousr-core + 'ousr-rest),$   
 $('event + 'iadv-core + 'iusr-core) + ('iadv-rest + 'iusr-rest),$   
 $(unit + 'oadv-core + 'ousr-core) + ('oadv-rest \times 'event\ list + 'ousr-rest \times$   
 $'event\ list))\ gpv$

**where**

*fuse-callee* (*Inl* (*Inl* *iadv-core*)) = *Pause* (*Inl* (*Inr* (*Inl* *iadv-core*))) ( $\lambda x.$  *case* *x* of  
*Inl* (*Inr* (*Inl* *oadv-core*))  $\Rightarrow$  *Done* (*Inl* (*Inl* *oadv-core*))  
| -  $\Rightarrow$  *Fail*)  
| *fuse-callee* (*Inl* (*Inr* *iadv-rest*)) = *Pause* (*Inr* (*Inl* *iadv-rest*)) ( $\lambda x.$  *case* *x* of  
*Inr* (*Inl* (*oadv-rest*, *es*))  $\Rightarrow$  *bind-gpv* (*pauses* (*map* (*Inl*  $\circ$  *Inl*) *es*)) ( $\lambda-$ . *Done*  
(*Inl* (*Inr* *oadv-rest*)))  
| -  $\Rightarrow$  *Fail*)  
| *fuse-callee* (*Inr* (*Inl* *iusr-core*)) = *Pause* (*Inl* (*Inr* (*Inr* *iusr-core*))) ( $\lambda x.$  *case* *x* of  
*Inl* (*Inr* (*Inr* *oadv-core*))  $\Rightarrow$  *Done* (*Inr* (*Inl* *oadv-core*)))  
| *fuse-callee* (*Inr* (*Inr* *iusr-rest*)) = *Pause* (*Inr* (*Inr* *iusr-rest*)) ( $\lambda x.$  *case* *x* of  
*Inr* (*Inr* (*ousr-rest*, *es*))  $\Rightarrow$  *bind-gpv* (*pauses* (*map* (*Inl*  $\circ$  *Inl*) *es*)) ( $\lambda-$ . *Done*  
(*Inr* (*Inr* *ousr-rest*))))

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

**definition** *fuse-converter*

$:: (('iadv-core + 'iadv-rest) + ('iusr-core + 'iusr-rest),$   
 $('oadv-core + 'oadv-rest) + ('ousr-core + 'ousr-rest),$   
 $('event + 'iadv-core + 'iusr-core) + ('iadv-rest + 'iusr-rest),$   
 $(unit + 'oadv-core + 'ousr-core) + ('oadv-rest \times 'event\ list + 'ousr-rest \times$   
 $'event\ list))\ converter$

**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*  $\triangleright$  (*resource-of-oracle* (*callee-of-core* *core*) *s-core*  $\parallel$  *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*  $\langle + \rangle$  *IA*  $\langle + \rangle$  *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*  $\langle + \rangle$  *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]*:

$(\mathcal{IAC} \oplus_{\mathcal{I}} \text{map-}\mathcal{I} \text{ id fst } \mathcal{IAR}) \oplus_{\mathcal{I}} (\mathcal{IUC} \oplus_{\mathcal{I}} \text{map-}\mathcal{I} \text{ id fst } \mathcal{IUR}), (\mathcal{IE} \oplus_{\mathcal{I}} (\mathcal{IAC} \oplus_{\mathcal{I}} \mathcal{IUC})) \oplus_{\mathcal{I}} (\mathcal{IAR} \oplus_{\mathcal{I}} \mathcal{IUR}) \vdash_C \text{fuse-converter } \checkmark$   
**if**  $\forall x. \forall (y, es) \in \text{responses-}\mathcal{I} \mathcal{IAR} x. \text{set } es \subseteq \text{outs-}\mathcal{I} \mathcal{IE} \forall x. \forall (y, es) \in \text{responses-}\mathcal{I} \mathcal{IUR} x. \text{set } es \subseteq \text{outs-}\mathcal{I} \mathcal{IE}$   
 ⟨*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-}\mathcal{I} \mathcal{IE}) (outs-}\mathcal{I} \mathcal{ICA}) (outs-}\mathcal{I} \mathcal{ICU}) (return-spmf s-core) (return-spmf s-core')*  
**and** *rest*: *trace-rest-eq rest1 rest2 (outs-}\mathcal{I} \mathcal{IRA}) (outs-}\mathcal{I} \mathcal{IRU}) (return-spmf s-rest) (return-spmf s-rest')*  
**and** *IC1*: *callee-invariant-on (callee-of-core core1) IC1 (\mathcal{IE} \oplus\_{\mathcal{I}} (\mathcal{ICA} \oplus\_{\mathcal{I}} \mathcal{ICU})) IC1 s-core*  
**and** *IC2*: *callee-invariant-on (callee-of-core core2) IC2 (\mathcal{IE} \oplus\_{\mathcal{I}} (\mathcal{ICA} \oplus\_{\mathcal{I}} \mathcal{ICU})) IC2 s-core'*  
**and** *IR1*: *callee-invariant-on (callee-of-rest rest1) IR1 (\mathcal{IRA} \oplus\_{\mathcal{I}} \mathcal{IRU}) IR1 s-rest*  
**and** *IR2*: *callee-invariant-on (callee-of-rest rest2) IR2 (\mathcal{IRA} \oplus\_{\mathcal{I}} \mathcal{IRU}) IR2 s-rest'*  
**and** *E1 [WT-intro]*:  $\forall x. \forall (y, es) \in \text{responses-}\mathcal{I} \mathcal{IRA} x. \text{set } es \subseteq \text{outs-}\mathcal{I} \mathcal{IE}$   
**and** *E2 [WT-intro]*:  $\forall x. \forall (y, es) \in \text{responses-}\mathcal{I} \mathcal{IRU} x. \text{set } es \subseteq \text{outs-}\mathcal{I} \mathcal{IE}$   
**shows** *trace-callee-eq (fused-resource.fuse core1 rest1) (fused-resource.fuse core2 rest2)*

rest2)  
 ((outs- $\mathcal{I}$  ICA <+> outs- $\mathcal{I}$  IRA) <+> (outs- $\mathcal{I}$  ICU <+> outs- $\mathcal{I}$  IRU))  
 (return-spmf (s-core, s-rest)) (return-spmf (s-core', s-rest'))  
 <proof>

**inductive** trace-eq-simcl :: ('s1 spmf  $\Rightarrow$  's2 spmf  $\Rightarrow$  bool)  $\Rightarrow$  's1 spmf  $\Rightarrow$  's2 spmf  
 $\Rightarrow$  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**  $\bigwedge x :: \text{nat. } x \in \text{set-spmf } p \implies S$  ( $f$   $x$ ) ( $g$   $x$ )

**lemma** trace-eq-simcl-bindI [intro?]: trace-eq-simcl  $S$  (bind-spmf  $p$   $f$ ) (bind-spmf  $p$   $g$ )  
 )  
**if**  $\bigwedge x. x \in \text{set-spmf } p \implies S$  ( $f$   $x$ ) ( $g$   $x$ )  
 <proof>

**lemma** trace-eq-simcl-bind: trace-eq-simcl  $S$  (bind-spmf  $p$   $f$ ) (bind-spmf  $p$   $g$ )  
**if** \*:  $\bigwedge x :: 'a. x \in \text{set-spmf } p \implies \text{trace-eq-simcl } S$  ( $f$   $x$ ) ( $g$   $x$ )  
 <proof>

**lemma** trace-eq-simcl-bind1-scale: trace-eq-simcl  $S$  (bind-spmf  $p$   $f$ ) (scale-spmf  
 (weight-spmf  $p$ )  $q$ )  
**if**  $\forall x \in \text{set-spmf } p. \text{trace-eq-simcl } S$  ( $f$   $x$ )  $q$   
 <proof>

**lemma** trace-eq-simcl-bind1: trace-eq-simcl  $S$  (bind-spmf  $p$   $f$ )  $q$   
**if**  $\forall x \in \text{set-spmf } p. \text{trace-eq-simcl } S$  ( $f$   $x$ )  $q$  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**  $\forall x \in \text{set-spmf } q. \text{trace-eq-simcl } S$   $p$  ( $f$   $x$ )  
 <proof>

**lemma** trace-eq-simcl-bind2: trace-eq-simcl  $S$   $p$  (bind-spmf  $q$   $f$ )  
**if**  $\forall x \in \text{set-spmf } q. \text{trace-eq-simcl } S$   $p$  ( $f$   $x$ ) 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  $\Rightarrow$  's2 spmf  $\Rightarrow$  bool  
 <proof>

**lemma** trace-eq-simcl-map: trace-eq-simcl  $S$  (map-spmf  $f$   $p$ ) (map-spmf  $g$   $p$ )  
**if**  $\forall x \in \text{set-spmf } p. S$  (return-spmf ( $f$   $x$ )) (return-spmf ( $g$   $x$ ))  
 <proof>

**lemma** *trace-eq-simcl-map1*: *trace-eq-simcl*  $S$  (*map-spmf*  $f$   $p$ )  $q$   
**if**  $\forall x \in \text{set-spmf } p. \text{trace-eq-simcl } S$  (*return-spmf* ( $f$   $x$ ))  $q$  *lossless-spmf*  $p$   
*<proof>*

**lemma** *trace-eq-simcl-map2*: *trace-eq-simcl*  $S$   $p$  (*map-spmf*  $f$   $q$ )  
**if**  $\forall x \in \text{set-spmf } q. \text{trace-eq-simcl } S$   $p$  (*return-spmf* ( $f$   $x$ )) *lossless-spmf*  $q$   
*<proof>*

**lemma** *trace-eq-simcl-return-spmf* [*simp*]: *trace-eq-simcl*  $S$  (*return-spmf*  $x$ ) (*return-spmf*  $y$ )  
 $\longleftrightarrow S$  (*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*:  $\bigwedge p \ q \ a. \llbracket S \ p \ q; a \in A \rrbracket \Longrightarrow$   
*bind-spmf*  $p$  ( $\lambda s. \text{map-spmf } \text{fst} \ (\text{callee1 } s \ a)$ ) = *bind-spmf*  $q$  ( $\lambda s. \text{map-spmf } \text{fst}$   
(*callee2*  $s \ a$ )  
**and** *sim*:  $\bigwedge p \ q \ a \ \text{res} \ b \ s'. \llbracket S \ p \ q; a \in A; \text{res} \in \text{set-spmf } q; (b, s') \in \text{set-spmf}$   
(*callee2*  $\text{res} \ a$ )  $\rrbracket$   
 $\Longrightarrow \text{trace-eq-simcl } S$  (*cond-spmf-fst* (*bind-spmf*  $p$  ( $\lambda s. \text{callee1 } s \ a$ ))  $b$ )  
(*cond-spmf-fst* (*bind-spmf*  $q$  ( $\lambda s. \text{callee2 } s \ a$ ))  $b$ )  
**and** *start*: *trace-eq-simcl*  $S$   $p \ q$  **and**  $a: a \in A$   
**shows** *trace-eq-simcl-callee-step*: *bind-spmf*  $p$  ( $\lambda s. \text{map-spmf } \text{fst} \ (\text{callee1 } s \ a)$ ) =  
*bind-spmf*  $q$  ( $\lambda s. \text{map-spmf } \text{fst} \ (\text{callee2 } s \ a)$ ) (**is** *?step*)  
**and** *trace-eq-simcl-callee-sim*:  $\bigwedge \text{res} \ b \ s'. \llbracket \text{res} \in \text{set-spmf } q; (b, s') \in \text{set-spmf}$   
(*callee2*  $\text{res} \ a$ )  $\rrbracket$   
 $\Longrightarrow \text{trace-eq-simcl } S$  (*cond-spmf-fst* (*bind-spmf*  $p$  ( $\lambda s. \text{callee1 } s \ a$ ))  $b$ )  
(*cond-spmf-fst* (*bind-spmf*  $q$  ( $\lambda s. \text{callee2 } s \ a$ ))  $b$ ) (**is**  $\bigwedge \text{res} \ b$   
 $s'. \llbracket ?\text{res } \text{res}; ?b \ \text{res} \ b \ s' \rrbracket \Longrightarrow ?\text{sim } \text{res} \ b \ s'$ )  
*<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*:  $\bigwedge p \ q \ a. \llbracket S \ p \ q; a \in A \rrbracket \Longrightarrow$   
*bind-spmf*  $p$  ( $\lambda s. \text{map-spmf } \text{fst} \ (\text{callee1 } s \ a)$ ) = *bind-spmf*  $q$  ( $\lambda s. \text{map-spmf } \text{fst}$   
(*callee2*  $s \ a$ )  
**and** *sim*:  $\bigwedge p \ q \ a \ \text{res} \ b \ s'. \llbracket S \ p \ q; a \in A; \text{res} \in \text{set-spmf } q; (b, s') \in \text{set-spmf}$   
(*callee2*  $\text{res} \ a$ )  $\rrbracket$   
 $\Longrightarrow \text{trace-eq-simcl } S$  (*cond-spmf-fst* (*bind-spmf*  $p$  ( $\lambda s. \text{callee1 } s \ a$ ))  $b$ )  
(*cond-spmf-fst* (*bind-spmf*  $q$  ( $\lambda s. \text{callee2 } s \ a$ ))  $b$ )  
**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**  $S :: 's\text{-core} \ \text{spmf} \Rightarrow 's\text{-core}' \ \text{spmf} \Rightarrow \text{bool}$   
**assumes** *start*:  $S \ p \ q$

**and** *step-cpoke*:  $\bigwedge p q e. \llbracket S p q; e \in E \rrbracket \implies$   
 $\text{weight-spmf } (\text{bind-spmf } p (\lambda s. \text{cpoke core1 } s e)) = \text{weight-spmf } (\text{bind-spmf } q$   
 $(\lambda s. \text{cpoke core2 } s e))$   
**and** *sim-cpoke*:  $\bigwedge p q e. \llbracket S p q; e \in E \rrbracket \implies$   
 $\text{trace-eq-simcl } S (\text{mk-lossless } (\text{bind-spmf } p (\lambda s. \text{cpoke core1 } s e))) (\text{mk-lossless}$   
 $(\text{bind-spmf } q (\lambda s. \text{cpoke core2 } s e)))$   
**and** *step-cfunc-adv*:  $\bigwedge p q ia. \llbracket S p q; ia \in IA \rrbracket \implies$   
 $\text{bind-spmf } p (\lambda s1. \text{map-spmf } \text{fst } (\text{cfunc-adv core1 } s1 ia)) = \text{bind-spmf } q (\lambda s2.$   
 $\text{map-spmf } \text{fst } (\text{cfunc-adv core2 } s2 ia))$   
**and** *sim-cfunc-adv*:  $\bigwedge p q ia s1 s2 s1' s2' oa. \llbracket S p q; ia \in IA;$   
 $s1 \in \text{set-spmf } p; s2 \in \text{set-spmf } q; (oa, s1') \in \text{set-spmf } (\text{cfunc-adv core1 } s1 ia);$   
 $(oa, s2') \in \text{set-spmf } (\text{cfunc-adv core2 } s2 ia) \rrbracket$   
 $\implies \text{trace-eq-simcl } S (\text{cond-spmf-fst } (\text{bind-spmf } p (\lambda s1. \text{cfunc-adv core1 } s1 ia))$   
 $oa) (\text{cond-spmf-fst } (\text{bind-spmf } q (\lambda s2. \text{cfunc-adv core2 } s2 ia)) oa)$   
**and** *step-cfunc-usr*:  $\bigwedge p q iu. \llbracket S p q; iu \in IU \rrbracket \implies$   
 $\text{bind-spmf } p (\lambda s1. \text{map-spmf } \text{fst } (\text{cfunc-usr core1 } s1 iu)) = \text{bind-spmf } q (\lambda s2.$   
 $\text{map-spmf } \text{fst } (\text{cfunc-usr core2 } s2 iu))$   
**and** *sim-cfunc-usr*:  $\bigwedge p q iu s1 s2 s1' s2' ou. \llbracket S p q; iu \in IU;$   
 $s1 \in \text{set-spmf } p; s2 \in \text{set-spmf } q; (ou, s1') \in \text{set-spmf } (\text{cfunc-usr core1 } s1 iu);$   
 $(ou, s2') \in \text{set-spmf } (\text{cfunc-usr core2 } s2 iu) \rrbracket$   
 $\implies \text{trace-eq-simcl } S (\text{cond-spmf-fst } (\text{bind-spmf } p (\lambda s1. \text{cfunc-usr core1 } s1 iu))$   
 $ou) (\text{cond-spmf-fst } (\text{bind-spmf } q (\lambda s2. \text{cfunc-usr core2 } s2 iu)) ou)$   
**shows** *trace-core-eq core1 core2 E IA IU p q*  
 $\langle \text{proof} \rangle$

## 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  $\times$  's-rest, 'event1, 'iadv-core + 'iadv-rest, 'iusr-core + 'iusr-rest, 'oadv-core  
+ 'oadv-rest, 'ousr-core + 'ousr-rest) *core*  
**where**  
 $\text{cpoke } \text{core-with-rest} = (\lambda (s\text{-core}, s\text{-rest}) e. \text{map-spmf } (\lambda s\text{-core}'. (s\text{-core}', s\text{-rest}))$   
 $(\text{cpoke } \text{core } s\text{-core } (\text{Inl } e)))$   
|  $\text{cfunc-adv } \text{core-with-rest} = (\lambda (s\text{-core}, s\text{-rest}) iadv. \text{case } iadv \text{ of}$   
 $\text{Inl } iadv\text{-core} \Rightarrow \text{map-spmf } (\lambda (oadv\text{-core}, s\text{-core}'). (\text{Inl } oadv\text{-core}, (s\text{-core}',$   
 $s\text{-rest}))) (\text{cfunc-adv } \text{core } s\text{-core } iadv\text{-core})$   
|  $\text{Inr } iadv\text{-rest} \Rightarrow$   
 $\text{bind-spmf } (\text{rfunc-adv } \text{rest } s\text{-rest } iadv\text{-rest}) (\lambda ((oadv\text{-rest}, es), s\text{-rest}').$   
 $\text{map-spmf } (\lambda s\text{-core}'. (\text{Inr } oadv\text{-rest}, (s\text{-core}', s\text{-rest}')))) (\text{foldl-spmf } (\text{cpoke}$   
 $\text{core}) (\text{return-spmf } s\text{-core}) (\text{map } \text{Inr } es)))$   
|  $\text{cfunc-usr } \text{core-with-rest} = (\lambda (s\text{-core}, s\text{-rest}) iusr. \text{case } iusr \text{ of}$   
 $\text{Inl } iusr\text{-core} \Rightarrow \text{map-spmf } (\lambda (ousr\text{-core}, s\text{-core}'). (\text{Inl } ousr\text{-core}, (s\text{-core}',$

```

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 (λs-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* ∘ (fst *si2*)) ((*o*) (snd *si2*)) *si1*) oracle

$\langle proof \rangle$

**lemma** *apply-wiring-state-iso-assoc*:

*apply-wiring wr (apply-state-iso si oracle) = apply-state-iso si (apply-wiring wr oracle)*

$\langle proof \rangle$

**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)*

$\langle proof \rangle$

## 6.1 Parallel State Isomorphism

**definition**

*parallel-state-iso* ::  $((s\text{-core1} \times s\text{-core2}) \times (s\text{-rest1} \times s\text{-rest2}), (s\text{-core1} \times s\text{-rest1}) \times (s\text{-core2} \times s\text{-rest2}))$  *state-iso*

**where**

*parallel-state-iso* =

$(\lambda((s11, s12), (s21, s22)). ((s11, s21), (s12, s22)), \lambda((s11, s21), (s12, s22)). ((s11, s12), (s21, s22)))$

**lemma**

*state-iso-parallel-state-iso [simp]: state-iso parallel-state-iso*

$\langle proof \rangle$

## 6.2 Trisplit State Isomorphism

**definition**

*iso-trisplit*

**where**

*iso-trisplit* =

$(\lambda(((s11, s12), s13), (s21, s22), s23)). (((s11, s21), s12, s22), s13, s23), \lambda(((s11, s21), s12, s22), s13, s23)). (((s11, s12), s13), (s21, s22), s23))$

**lemma**

*state-iso-fuse-par [simp]: state-iso iso-trisplit*

$\langle proof \rangle$

## 6.3 Assoc-Swap State Isomorphism

**definition**

*iso-swapar*

**where**

*iso-swapar* =  $(\lambda((sm, s1), s2). (s1, sm, s2), \lambda(s1, sm, s2). ((sm, s1), s2))$

**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) = (λinp. 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 (λc. *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) = (λinp. case inp of  
    Inl x ⇒ map-gpv (map-prod Inl (λc. *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* (λs q. case-sum (λql. Done (Inl (f ql), s)) (λqr. 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* (λs q. case-sum (λql. map-gpv (map-prod Inl id) id (callee s  
  ql)) (λqr. 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**

$wiring\text{-}c1r22\text{-}c1r22 :: ('icnv\text{-}usr1 + 'icnv\text{-}usr2, 'ocnv\text{-}usr1 + 'ocnv\text{-}usr2,$   
 $('iusr1\text{-}res1 + 'iusr2\text{-}res1) + 'iusr1\text{-}res2 + 'iusr2\text{-}res2,$   
 $('ousr1\text{-}res1 + 'ousr2\text{-}res1) + 'ousr1\text{-}res2 + 'ousr2\text{-}res2) \text{ converter}$

**where**

$wiring\text{-}c1r22\text{-}c1r22 \equiv (c1r22 \mid = c1r22) \odot \text{parallel-wiring}$

**end**

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

**definition**

$fused\text{-}wiring ::$   
 $((('iadv\text{-}core1 + 'iadv\text{-}core2) + ('iadv\text{-}rest1 + 'iadv\text{-}rest2)) +$   
 $(('iusr\text{-}core1 + 'iusr\text{-}core2) + ('iusr\text{-}rest1 + 'iusr\text{-}rest2)),$   
 $(('oadv\text{-}core1 + 'oadv\text{-}core2) + ('oadv\text{-}rest1 + 'oadv\text{-}rest2)) +$   
 $(('ousr\text{-}core1 + 'ousr\text{-}core2) + ('ousr\text{-}rest1 + 'ousr\text{-}rest2)),$   
 $(('iadv\text{-}core1 + 'iadv\text{-}rest1) + ('iusr\text{-}core1 + 'iusr\text{-}rest1)) +$   
 $(('iadv\text{-}core2 + 'iadv\text{-}rest2) + ('iusr\text{-}core2 + 'iusr\text{-}rest2)),$   
 $(('oadv\text{-}core1 + 'oadv\text{-}rest1) + ('ousr\text{-}core1 + 'ousr\text{-}rest1)) +$   
 $(('oadv\text{-}core2 + 'oadv\text{-}rest2) + ('ousr\text{-}core2 + 'ousr\text{-}rest2))) \text{ converter}$

**where**

$fused\text{-}wiring \equiv (\text{parallel-wiring} \mid = \text{parallel-wiring}) \odot \text{parallel-wiring}$

**definition**

$fused\text{-}wiring_w$

**where**

$fused\text{-}wiring_w \equiv (\text{parallel-wiring}_w \mid_w \text{parallel-wiring}_w) \circ_w \text{parallel-wiring}_w$

**schematic-goal**

$wiring\text{-}fused\text{-}wiring[wiring\text{-}intro]: wiring \ ?I1 \ ?I2 \ fused\text{-}wiring \ fused\text{-}wiring_w$   
 $\langle \text{proof} \rangle$

**schematic-goal**  $WT\text{-}fused\text{-}wiring [WT\text{-}intro]: \ ?I1, \ ?I2 \vdash_C \ fused\text{-}wiring \ \checkmark$   
 $\langle \text{proof} \rangle$

**context**

**fixes**

$c1r22 :: ('icnv\text{-}usr1, 'ocnv\text{-}usr1, 'iusr1\text{-}core1 + 'iusr1\text{-}core2, 'ousr1\text{-}core1 +$   
 $'ousr1\text{-}core2) \text{ converter}$  **and**

$c1r22 :: ('icnv\text{-}usr2, 'ocnv\text{-}usr2, 'iusr2\text{-}core1 + 'iusr2\text{-}core2, 'ousr2\text{-}core1 +$   
 $'ousr2\text{-}core2) \text{ converter}$  **and**

$res1 :: (('iadv\text{-}core1 + 'iadv\text{-}rest1) + ('iusr1\text{-}core1 + 'iusr2\text{-}core1) + 'iusr\text{-}rest1,$

```

      ('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

```

— Attachment 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 = (((1C |= 1C) |= ((wiring-c1r22-c1r22 cnv1 cnv2) |= 1C))
  ⊙ 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-wiringw (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* = ( $\lambda$ (*s-event*, *s-core*) *event*).

*bind-spmf* (*etran s-event event*) ( $\lambda$ (*events*, *s-event*).

*map-spmf* ( $\lambda$ *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* *I-adv* *I-usr* *I-rest rest*

**shows** *WT-rest* *I-adv* *I-usr* (*pred-prod* ( $\lambda$ -. *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**  $cpoke\ core' = (\lambda s. case-sum\ (\lambda q. fn\ s\ q)\ (cpoke\ core\ s))$   
**and**  $cfunc-adv\ core = cfunc-adv\ core'$   
**and**  $cfunc-usr\ core = cfunc-usr\ core' \oplus_O (\lambda s\ i. map-spmf\ (Pair\ (h-out\ i))\ (fn\ s\ i))$   
**and**  $rfunc-adv\ rest' = (\lambda s\ q. map-spmf\ (apfst\ (apsnd\ (map\ Inr))))\ (rfunc-adv\ rest\ s\ q)$   
**and**  $rfunc-usr\ rest' = plus-eoracle\ (\lambda s\ i. return-spmf\ ((h-out\ i, [i]), s))\ (rfunc-usr\ rest)$   
**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**  $x-ifunc \equiv (\lambda ifunc\ core\ (se, sc)\ q. do\ \{$   
   $((out, es), se') \leftarrow ifunc\ se\ q;$   
   $sc' \leftarrow foldl-spmf\ (cpoke\ core)\ (return-spmf\ sc)\ es;$   
   $return-spmf\ (out, se', sc')\ \}$   
**assumes**  $cpoke\ core' = cpoke\ (translate-core\ etran\ core)$   
**and**  $cfunc-adv\ core' = \dagger(cfunc-adv\ core) \oplus_O x-ifunc\ ifunc\ core$   
**and**  $cfunc-usr\ core' = cfunc-usr\ (translate-core\ etran\ core)$   
**and**  $rinit\ rest = (einit, rinit\ rest')$   
**and**  $rfunc-adv\ rest = (\lambda s\ q. case\ q\ of$   
   $Inl\ ql \Rightarrow map-spmf\ (apfst\ (map-prod\ Inl\ id))\ ((ifunc\dagger)\ s\ ql)$   
   $| Inr\ qr \Rightarrow map-spmf\ (apfst\ (map-prod\ Inr\ id))\ ((translate-eoracle\ etran$   
 $(\dagger(rfunc-adv\ rest')))\ s\ qr))$   
**and**  $rfunc-usr\ rest = translate-eoracle\ etran\ (\dagger(rfunc-usr\ rest'))$   
**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** *I-real* :: ('a, 'b)  $\mathcal{I}$   
**and** *I-ideal* :: ('c, 'd)  $\mathcal{I}$   
**and** *I-common* :: ('e, 'f)  $\mathcal{I}$   
**and** *adv* :: real  
**assumes** *WT-real* [*WT-intro*]:  $\mathcal{I}\text{-real} \oplus_{\mathcal{I}} \mathcal{I}\text{-common} \vdash_{\text{res}} \text{real-resource} \checkmark$   
**and** *WT-ideal* [*WT-intro*]:  $\mathcal{I}\text{-ideal} \oplus_{\mathcal{I}} \mathcal{I}\text{-common} \vdash_{\text{res}} \text{ideal-resource} \checkmark$   
**and** *WT-sim* [*WT-intro*]:  $\mathcal{I}\text{-real}, \mathcal{I}\text{-ideal} \vdash_C \text{sim} \checkmark$   
**and** *pfinite-sim* [*pfinite-intro*]: *pfinite-converter*  $\mathcal{I}\text{-real} \mathcal{I}\text{-ideal} \text{sim}$   
**and** *adv-nonneg*:  $0 \leq \text{adv}$

**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** *I-real* :: ('a, 'b)  $\mathcal{I}$   
**and** *I-common* :: ('e, 'f)  $\mathcal{I}$   
**and** *A* :: ('a + 'e, 'b + 'f) distinguisher-obsf  
**and** *adv* :: real  
**assumes** *adv*: [ *exception-I* ( $\mathcal{I}\text{-real} \oplus_{\mathcal{I}} \mathcal{I}\text{-common}$ )  $\vdash_g A \checkmark$  ]  
 $\implies \text{advantage } A (\text{obsf-resource } (\text{sim} \mid= 1_C \triangleright \text{ideal-resource})) (\text{obsf-resource } (\text{real-resource})) \leq \text{adv}$

**locale** *constructive-security-obsf* = *constructive-security-aux-obsf* *real-resource* *ideal-resource*  
*sim* *I-real* *I-ideal* *I-common* *adv*  
+ *constructive-security-sim-obsf* *real-resource* *ideal-resource* *sim* *I-real* *I-common*  
*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** *I-real* :: ('a, 'b)  $\mathcal{I}$   
**and** *I-ideal* :: ('c, 'd)  $\mathcal{I}$   
**and** *I-common* :: ('e, 'f)  $\mathcal{I}$   
**and** *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* *I-real* *I-ideal* *I-common* *adv*  $\langle \text{proof} \rangle$   
**lemma** *constructive-security-sim-obsf*: *constructive-security-sim-obsf* *real-resource* *ideal-resource* *sim* *I-real* *I-common* *A* *adv*  $\langle \text{proof} \rangle$

**end**

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

**lemma** *constructive-security-obsf-refl*:

*constructive-security-obsf real-resource ideal-resource sim I-real I-ideal I-common*  
*A*  
(*advantage A (obsf-resource (sim |<sub>=</sub> 1<sub>C</sub> ▷ ideal-resource)) (obsf-resource*  
*(real-resource))*)  
{*proof*}

**end**

**lemma** *constructive-security-obsf-absorb-cong*:

**assumes** *sec: constructive-security-obsf real-resource ideal-resource sim I-real*  
*I-ideal I-common (absorb A cnv) adv*  
**and** [*WT-intro*]: *exception-I I, exception-I (I-real ⊕<sub>I</sub> I-common) ⊢<sub>C</sub> cnv √*  
*exception-I I, exception-I (I-real ⊕<sub>I</sub> I-common) ⊢<sub>C</sub> cnv' √ exception-I I ⊢<sub>g</sub> A*  
*√*  
**and** *cong: exception-I I, exception-I (I-real ⊕<sub>I</sub> I-common) ⊢<sub>C</sub> cnv ~ cnv'*  
**shows** *constructive-security-obsf real-resource ideal-resource sim I-real I-ideal*  
*I-common (absorb A cnv') adv*  
{*proof*}

**lemma** *constructive-security-obsf-sim-cong*:

**assumes** *sec: constructive-security-obsf real-resource ideal-resource sim I-real*  
*I-ideal I-common A adv*  
**and** *cong: I-real, I-ideal ⊢<sub>C</sub> sim ~ sim'*  
**and** *pfinite [pfinite-intro]: pfinite-converter I-real I-ideal sim'*  
**shows** *constructive-security-obsf real-resource ideal-resource sim' I-real I-ideal*  
*I-common A adv*  
{*proof*}

**lemma** *constructive-security-obsfI-core-rest [locale-witness]*:

**assumes** *constructive-security-aux-obsf real-resource ideal-resource sim I-real*  
*I-ideal (I-common-core ⊕<sub>I</sub> I-common-rest) adv*  
**and** *adv: [ [ exception-I (I-real ⊕<sub>I</sub> (I-common-core ⊕<sub>I</sub> I-common-rest)) ⊢<sub>g</sub> A*  
*√ ] ]*  
 $\implies$  *advantage A (obsf-resource (sim |<sub>=</sub> (1<sub>C</sub> |<sub>=</sub> 1<sub>C</sub>) ▷ ideal-resource))*  
*(obsf-resource (real-resource)) ≤ adv*  
**shows** *constructive-security-obsf real-resource ideal-resource sim I-real I-ideal*  
*(I-common-core ⊕<sub>I</sub> I-common-rest) 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  $\models 1_C$ ))) 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')  $\vdash_g \mathcal{A} \checkmark$*   
 $\implies$  *constructive-security-sim-obsf real-resource ideal-resource sim  $\mathcal{I}$ -real  $\mathcal{I}$ -common*  
*(absorb  $\mathcal{A}$  (obsf-converter (w-adv-real  $\models$  w-usr))) adv*  
 (is  $\implies$  *constructive-security-sim-obsf - - - - ? $\mathcal{A}$  -*)  
**assumes** *WT-usr [WT-intro]:  $\mathcal{I}$ -common',  $\mathcal{I}$ -common  $\vdash_C$  w-usr  $\checkmark$*   
**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  $\checkmark$*   
**and** *WT-w-adv-ideal [WT-intro]:  $\mathcal{I}$ -ideal',  $\mathcal{I}$ -ideal  $\vdash_C$  w-adv-ideal  $\checkmark$*   
**and** *WT-adv-ideal-inv [WT-intro]:  $\mathcal{I}$ -ideal,  $\mathcal{I}$ -ideal'  $\vdash_C$  w-adv-ideal-inv  $\checkmark$*   
**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  $\models$  w-usr  $\triangleright$  real-resource) (w-adv-ideal*  
 *$\models$  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  $\models$  w-usr))) adv*  
**assumes** *WT-usr [WT-intro]:  $\mathcal{I}$ -common',  $\mathcal{I}$ -common  $\vdash_C$  w-usr  $\checkmark$*   
**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  $\checkmark$*   
**and** *WT-w-adv-ideal [WT-intro]:  $\mathcal{I}$ -ideal',  $\mathcal{I}$ -ideal  $\vdash_C$  w-adv-ideal  $\checkmark$*   
**and** *WT-adv-ideal-inv [WT-intro]:  $\mathcal{I}$ -ideal,  $\mathcal{I}$ -ideal'  $\vdash_C$  w-adv-ideal-inv  $\checkmark$*   
**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  $\models$  w-usr  $\triangleright$  real-resource) (w-adv-ideal*  
 *$\models$  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')  $\vdash_g \mathcal{A} \checkmark$*

$\implies$  *constructive-security-sim-obsf* *real-resource* *ideal-resource* *sim*  $\mathcal{I}$ -*real*  $\mathcal{I}$ -*common*  
 (absorb  $\mathcal{A}$  (obsf-converter ( $1_C \models conv$ ))) adv  
 and *WT-conv* [*WT-intro*]:  $\mathcal{I}$ -*common'*,  $\mathcal{I}$ -*common*  $\vdash_C conv \checkmark$   
 and *pfinite* [*pfinite-intro*]: *pfinite-converter*  $\mathcal{I}$ -*common'*  $\mathcal{I}$ -*common* *conv*  
 shows *constructive-security-obsf* ( $1_C \models conv \triangleright real-resource$ ) ( $1_C \models conv \triangleright ideal-resource$ ) *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'*)  $\vdash_g \mathcal{A} \checkmark$   
 $\implies$  *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 (( $1_C \models 1_C \models conv$ )))) adv  
 assumes *WT-conv* [*WT-intro*]:  $\mathcal{I}$ -*common'*,  $\mathcal{I}$ -*common*  $\vdash_C conv \checkmark$   
 and *pfinite* [*pfinite-intro*]: *pfinite-converter*  $\mathcal{I}$ -*common'*  $\mathcal{I}$ -*common* *conv*  
 shows *constructive-security-obsf* (( $1_C \models 1_C \models conv \triangleright real-resource$ ) (( $1_C \models 1_C \models conv \triangleright ideal-resource$ )) *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*  $\vdash_{res} res \checkmark$   
 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*  $\models$  *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*  $\models$   $1_C \triangleright ideal2$ )))))) *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*

▷  $ideal1 \parallel ideal2$  ( $sim1 \models sim2$ )  
 ( $\mathcal{I}\text{-real1} \oplus_{\mathcal{I}} \mathcal{I}\text{-real2}$ ) ( $\mathcal{I}\text{-inner1} \oplus_{\mathcal{I}} \mathcal{I}\text{-inner2}$ ) ( $\mathcal{I}\text{-common1} \oplus_{\mathcal{I}} \mathcal{I}\text{-common2}$ )  
 $\mathcal{A}$  ( $adv1 + adv2$ )  
 ⟨proof⟩

**theorem** *parallel-constructive-security-obsf-fuse*:

**assumes** 1: *constructive-security-obsf*  $real1$   $ideal1$   $sim1$  ( $\mathcal{I}\text{-real1-core} \oplus_{\mathcal{I}} \mathcal{I}\text{-real1-rest}$ )  
 ( $\mathcal{I}\text{-ideal1-core} \oplus_{\mathcal{I}} \mathcal{I}\text{-ideal1-rest}$ ) ( $\mathcal{I}\text{-common1-core} \oplus_{\mathcal{I}} \mathcal{I}\text{-common1-rest}$ ) (*absorb*  $\mathcal{A}$   
 (*obsf-converter* (*fused-wiring*  $\odot$  *parallel-converter*  $1_C$  (*converter-of-resource* ( $sim2$   
 $\models 1_C \triangleright ideal2$ ))))))  $adv1$

(*is constructive-security-obsf* - - - ? $\mathcal{I}\text{-real1}$  ? $\mathcal{I}\text{-ideal1}$  ? $\mathcal{I}\text{-common1}$  ? $\mathcal{A}1$  -)

**assumes** 2: *constructive-security-obsf*  $real2$   $ideal2$   $sim2$  ( $\mathcal{I}\text{-real2-core} \oplus_{\mathcal{I}} \mathcal{I}\text{-real2-rest}$ )  
 ( $\mathcal{I}\text{-ideal2-core} \oplus_{\mathcal{I}} \mathcal{I}\text{-ideal2-rest}$ ) ( $\mathcal{I}\text{-common2-core} \oplus_{\mathcal{I}} \mathcal{I}\text{-common2-rest}$ ) (*absorb*  $\mathcal{A}$   
 (*obsf-converter* (*fused-wiring*  $\odot$  *parallel-converter* (*converter-of-resource*  $real1$ )  
 $1_C$ )))  $adv2$

(*is constructive-security-obsf* - - - ? $\mathcal{I}\text{-real2}$  ? $\mathcal{I}\text{-ideal2}$  ? $\mathcal{I}\text{-common2}$  ? $\mathcal{A}2$  -)

**shows** *constructive-security-obsf* (*fused-wiring*  $\triangleright real1 \parallel real2$ ) (*fused-wiring*  $\triangleright$   
 $ideal1 \parallel ideal2$ )

(*parallel-wiring*  $\odot$  ( $sim1 \models sim2$ )  $\odot$  *parallel-wiring*)

(( $\mathcal{I}\text{-real1-core} \oplus_{\mathcal{I}} \mathcal{I}\text{-real2-core}$ )  $\oplus_{\mathcal{I}}$  ( $\mathcal{I}\text{-real1-rest} \oplus_{\mathcal{I}} \mathcal{I}\text{-real2-rest}$ ))

(( $\mathcal{I}\text{-ideal1-core} \oplus_{\mathcal{I}} \mathcal{I}\text{-ideal2-core}$ )  $\oplus_{\mathcal{I}}$  ( $\mathcal{I}\text{-ideal1-rest} \oplus_{\mathcal{I}} \mathcal{I}\text{-ideal2-rest}$ ))

(( $\mathcal{I}\text{-common1-core} \oplus_{\mathcal{I}} \mathcal{I}\text{-common2-core}$ )  $\oplus_{\mathcal{I}}$  ( $\mathcal{I}\text{-common1-rest} \oplus_{\mathcal{I}} \mathcal{I}\text{-common2-rest}$ ))

$\mathcal{A}$  ( $adv1 + adv2$ )

⟨proof⟩

**end**

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

## 8 Asymptotic security definition

**locale** *constructive-security-obsf'* =

**fixes**  $real\text{-resource} :: security \Rightarrow ('a + 'e, 'b + 'f) resource$

**and**  $ideal\text{-resource} :: security \Rightarrow ('c + 'e, 'd + 'f) resource$

**and**  $sim :: security \Rightarrow ('a, 'b, 'c, 'd) converter$

**and**  $\mathcal{I}\text{-real} :: security \Rightarrow ('a, 'b) \mathcal{I}$

**and**  $\mathcal{I}\text{-ideal} :: security \Rightarrow ('c, 'd) \mathcal{I}$

**and**  $\mathcal{I}\text{-common} :: security \Rightarrow ('e, 'f) \mathcal{I}$

**and**  $\mathcal{A} :: security \Rightarrow ('a + 'e, 'b + 'f) distinguisher\text{-obsf}$

**assumes** *constructive-security-aux-obsf*:  $\bigwedge \eta.$

*constructive-security-aux-obsf* ( $real\text{-resource} \eta$ ) ( $ideal\text{-resource} \eta$ ) ( $sim \eta$ ) ( $\mathcal{I}\text{-real}$   
 $\eta$ ) ( $\mathcal{I}\text{-ideal} \eta$ ) ( $\mathcal{I}\text{-common} \eta$ ) 0

**and**  $adv: \llbracket \bigwedge \eta. exception\text{-}\mathcal{I} (\mathcal{I}\text{-real} \eta \oplus_{\mathcal{I}} \mathcal{I}\text{-common} \eta) \vdash_g \mathcal{A} \eta \checkmark \rrbracket$

$\implies negligible (\lambda \eta. advantage (\mathcal{A} \eta) (obsf\text{-resource} (sim \eta \models 1_C \triangleright ideal\text{-resource}$   
 $\eta)) (obsf\text{-resource} (real\text{-resource} \eta)))$

**begin**

**sublocale** *constructive-security-aux-obsf*

$real\text{-resource} \eta$

$ideal\text{-resource} \eta$

$\text{sim } \eta$   
 $\mathcal{I}\text{-real } \eta$   
 $\mathcal{I}\text{-ideal } \eta$   
 $\mathcal{I}\text{-common } \eta$   
 $0$   
**for**  $\eta$   $\langle \text{proof} \rangle$

**lemma** *constructive-security-obsf'D:*

$\text{constructive-security-obsf } (\text{real-resource } \eta) (\text{ideal-resource } \eta) (\text{sim } \eta) (\mathcal{I}\text{-real } \eta)$   
 $(\mathcal{I}\text{-ideal } \eta) (\mathcal{I}\text{-common } \eta) (\mathcal{A} \eta)$   
 $(\text{advantage } (\mathcal{A} \eta) (\text{obsf-resource } (\text{sim } \eta \mid = 1_C \triangleright \text{ideal-resource } \eta)) (\text{obsf-resource } (\text{real-resource } \eta))))$   
 $\langle \text{proof} \rangle$

**end**

**lemma** *constructive-security-obsf'I:*

**assumes**  $\bigwedge \eta. \text{constructive-security-obsf } (\text{real-resource } \eta) (\text{ideal-resource } \eta) (\text{sim } \eta)$   
 $(\mathcal{I}\text{-real } \eta) (\mathcal{I}\text{-ideal } \eta) (\mathcal{I}\text{-common } \eta) (\mathcal{A} \eta) (\text{adv } \eta)$   
**and**  $(\bigwedge \eta. \text{exception-}\mathcal{I} (\mathcal{I}\text{-real } \eta \oplus_{\mathcal{I}} \mathcal{I}\text{-common } \eta) \vdash_g \mathcal{A} \eta \checkmark) \implies \text{negligible adv}$   
**shows**  $\text{constructive-security-obsf}' \text{ real-resource ideal-resource sim } \mathcal{I}\text{-real } \mathcal{I}\text{-ideal}$   
 $\mathcal{I}\text{-common } \mathcal{A}$   
 $\langle \text{proof} \rangle$

**lemma** *constructive-security-obsf'-into-constructive-security:*

**assumes**  $\bigwedge \mathcal{A} :: \text{security} \implies ('a + 'b, 'c + 'd) \text{distinguisher-obsf.}$   
 $\llbracket \bigwedge \eta. \text{interaction-bounded-by } (\lambda-. \text{True}) (\mathcal{A} \eta) (\text{bound } \eta);$   
 $\bigwedge \eta. \text{lossless} \implies \text{plossless-gpv } (\text{exception-}\mathcal{I} (\mathcal{I}\text{-real } \eta \oplus_{\mathcal{I}} \mathcal{I}\text{-common } \eta)) (\mathcal{A} \eta)$   
 $\rrbracket$   
 $\implies \text{constructive-security-obsf}' \text{ real-resource ideal-resource sim } \mathcal{I}\text{-real } \mathcal{I}\text{-ideal}$   
 $\mathcal{I}\text{-common } \mathcal{A}$   
**and**  $\text{correct: } \exists \text{cnv. } \forall \mathcal{D}. (\forall \eta. \mathcal{I}\text{-ideal } \eta \oplus_{\mathcal{I}} \mathcal{I}\text{-common } \eta \vdash_g \mathcal{D} \eta \checkmark) \longrightarrow$   
 $(\forall \eta. \text{interaction-any-bounded-by } (\mathcal{D} \eta) (\text{bound } \eta)) \longrightarrow$   
 $(\forall \eta. \text{lossless} \longrightarrow \text{plossless-gpv } (\mathcal{I}\text{-ideal } \eta \oplus_{\mathcal{I}} \mathcal{I}\text{-common } \eta) (\mathcal{D} \eta)) \longrightarrow$   
 $(\forall \eta. \text{wiring } (\mathcal{I}\text{-ideal } \eta) (\mathcal{I}\text{-real } \eta) (\text{cnv } \eta) (w \eta)) \wedge$   
 $\text{Negligible.negligible } (\lambda \eta. \text{advantage } (\mathcal{D} \eta) (\text{ideal-resource } \eta) (\text{cnv } \eta \mid =$   
 $1_C \triangleright \text{real-resource } \eta))$   
**shows**  $\text{constructive-security real-resource ideal-resource sim } \mathcal{I}\text{-real } \mathcal{I}\text{-ideal } \mathcal{I}\text{-common}$   
 $\text{bound lossless } w$   
 $\langle \text{proof} \rangle$

## 8.1 Composition theorems

**theorem** *constructive-security-obsf'-composability:*

**fixes**  $\text{real}$   
**assumes**  $\text{constructive-security-obsf}' \text{ middle ideal sim-inner } \mathcal{I}\text{-middle } \mathcal{I}\text{-inner}$   
 $\mathcal{I}\text{-common } (\lambda \eta. \text{absorb } (\mathcal{A} \eta) (\text{obsf-converter } (\text{sim-outer } \eta \mid = 1_C)))$   
**assumes**  $\text{constructive-security-obsf}' \text{ real middle sim-outer } \mathcal{I}\text{-real } \mathcal{I}\text{-middle } \mathcal{I}\text{-common}$   
 $\mathcal{A}$

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

**theorem** *constructive-security-obsf'-lifting*:

**assumes** *sec*: *constructive-security-obsf'* real-resource ideal-resource sim  $\mathcal{I}$ -real  
 $\mathcal{I}$ -ideal  $\mathcal{I}$ -common ( $\lambda\eta. \text{absorb } (\mathcal{A} \ \eta) \ (\text{obsf-converter } (1_C \mid = \text{conv } \eta))$ )

**assumes** *WT-conv* [*WT-intro*]:  $\bigwedge\eta. \mathcal{I}$ -common'  $\eta, \mathcal{I}$ -common  $\eta \vdash_C \text{conv } \eta \ \checkmark$

**and** *pfinite* [*pfinite-intro*]:  $\bigwedge\eta. \text{pfinite-converter } (\mathcal{I}$ -common'  $\eta) \ (\mathcal{I}$ -common  $\eta)$   
 $(\text{conv } \eta)$

**shows** *constructive-security-obsf'*

( $\lambda\eta. 1_C \mid = \text{conv } \eta \triangleright \text{real-resource } \eta$ ) ( $\lambda\eta. 1_C \mid = \text{conv } \eta \triangleright \text{ideal-resource } \eta$ ) sim  
 $\mathcal{I}$ -real  $\mathcal{I}$ -ideal  $\mathcal{I}$ -common'  $\mathcal{A}$

$\langle \text{proof} \rangle$

**theorem** *constructive-security-obsf'-trivial*:

**fixes** *res*

**assumes** [*WT-intro*]:  $\bigwedge\eta. \mathcal{I} \ \eta \oplus_{\mathcal{I}} \mathcal{I}$ -common  $\eta \vdash_{\text{res}} \text{res } \eta \ \checkmark$

**shows** *constructive-security-obsf'* res res ( $\lambda-. 1_C$ )  $\mathcal{I} \ \mathcal{I}$   $\mathcal{I}$ -common  $\mathcal{A}$

$\langle \text{proof} \rangle$

**theorem** *parallel-constructive-security-obsf'*:

**assumes** *constructive-security-obsf'* real1 ideal1 sim1  $\mathcal{I}$ -real1  $\mathcal{I}$ -inner1  $\mathcal{I}$ -common1  
 $(\lambda\eta. \text{absorb } (\mathcal{A} \ \eta) \ (\text{obsf-converter } (\text{parallel-wiring } \odot \text{parallel-converter } 1_C \ (\text{converter-of-resource}$   
 $(\text{sim2 } \eta \mid = 1_C \triangleright \text{ideal2 } \eta))))$ )

(**is** *constructive-security-obsf'* - - - - - ? $\mathcal{A}1$ )

**assumes** *constructive-security-obsf'* real2 ideal2 sim2  $\mathcal{I}$ -real2  $\mathcal{I}$ -inner2  $\mathcal{I}$ -common2

$(\lambda\eta. \text{absorb } (\mathcal{A} \ \eta) \ (\text{obsf-converter } (\text{parallel-wiring } \odot \text{parallel-converter } (\text{converter-of-resource}$   
 $(\text{real1 } \eta)) \ 1_C))))$

(**is** *constructive-security-obsf'* - - - - - ? $\mathcal{A}2$ )

**shows** *constructive-security-obsf'* ( $\lambda\eta. \text{parallel-wiring } \triangleright \text{real1 } \eta \parallel \text{real2 } \eta$ ) ( $\lambda\eta.$   
 $\text{parallel-wiring } \triangleright \text{ideal1 } \eta \parallel \text{ideal2 } \eta$ ) ( $\lambda\eta. \text{sim1 } \eta \mid = \text{sim2 } \eta$ )

( $\lambda\eta. \mathcal{I}$ -real1  $\eta \oplus_{\mathcal{I}} \mathcal{I}$ -real2  $\eta$ ) ( $\lambda\eta. \mathcal{I}$ -inner1  $\eta \oplus_{\mathcal{I}} \mathcal{I}$ -inner2  $\eta$ ) ( $\lambda\eta. \mathcal{I}$ -common1  
 $\eta \oplus_{\mathcal{I}} \mathcal{I}$ -common2  $\eta$ )  $\mathcal{A}$

$\langle \text{proof} \rangle$

**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*  $\times$  *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*]:  $Alice \neq x \longleftrightarrow x = Bob$   
*<proof>*

**lemma** *neq-Alice-iff* [*simp*]:  $x \neq Alice \longleftrightarrow x = Bob$   
*<proof>*

**lemma** *Bob-neq-iff* [*simp*]:  $Bob \neq x \longleftrightarrow x = Alice$   
*<proof>*

**lemma** *neq-Bob-iff* [*simp*]:  $x \neq Bob \longleftrightarrow x = Alice$   
*<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>*

## 9.2 Defining the event handler

**fun** *poke* :: (*'key state, event*) *handler*

**where**

*poke* (*s-kernel, parties*) (*Event-Shell party*) =  
  (*if party*  $\in$  *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' ∈ set-spmf (poke s x) ↔
(∃ s-kernel parties party. s = (s-kernel, parties) ∧ x = Event-Shell party ∧ party
∉ parties ∧ s' = (s-kernel, insert party parties)) ∨
(∃ s-shell key. s = (PState-Store, s-shell) ∧ x = Event-Kernel ∧ key ∈ valid-keys
∧ finite valid-keys ∧ s' = (State-Store key, s-shell))
⟨proof⟩

```

**lemma** *foldl-poke-invar*:

```

[[ (s-kernel', parties') ∈ set-spmf (foldl-spmf poke p events); ∀(s-kernel, parties)
∈ set-spmf p. set-s-kernel s-kernel ⊆ valid-keys ]]
⇒ set-s-kernel s-kernel' ⊆ 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*  $\equiv$  *plus-oracle iface-alice iface-bob*

**lemma** *in-set-iface-usr-func* [*simp*]:

$x \in \text{set-spmf } (\text{iface-usr-func } \text{party } \text{wrap } \text{state } \text{inp}) \longleftrightarrow$

$(\exists \text{key } \text{parties. } \text{state} = (\text{State-Store } \text{key}, \text{parties}) \wedge \text{party} \in \text{parties} \wedge x = (\text{wrap } \text{key}, \text{State-Store } \text{key}, \text{parties}))$

$\langle \text{proof} \rangle$

**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*, {})  $\langle \text{proof} \rangle$

### 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))

$(\text{pred-prod } (\text{pred-s-kernel } (\lambda \text{key. } \text{key} \in \text{valid-keys})) (\lambda \cdot \text{True})) \text{ core}$

$\langle \text{proof} \rangle$

**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)  $\vdash_{\text{res}}$  *resource rest*  $\checkmark$

$\langle \text{proof} \rangle$

**end**

**end**

**theory** *Channel*

**imports**

*../Fused-Resource*

**begin**

## 10 Channel specification

**locale** *ideal-channel* =

**fixes**

```

    leak :: 'msg  $\Rightarrow$  '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  $\times$  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  $\neq$  x  $\longleftrightarrow$  x = Bob
  <proof>

```

```

lemma neq-Alice-iff [simp]: x  $\neq$  Alice  $\longleftrightarrow$  x = Bob
  <proof>

```

```

lemma Bob-neq-iff [simp]: Bob  $\neq$  x  $\longleftrightarrow$  x = Alice
  <proof>

```

```

lemma neq-Bob-iff [simp]: x  $\neq$  Bob  $\longleftrightarrow$  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* =  $(\lambda(s, ps) e. \text{map-spmf } (Pair\ s) (\text{case } e \text{ of Event-Shell party} \Rightarrow \text{if party} \in ps \text{ then return-pmf None else return-spmf (insert party ps)}))$   
 ⟨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*:  $ys' \in \text{set-spmf } (iface-drop\ s\ x) \longleftrightarrow \text{False}$   
 ⟨proof⟩

**lemma** *in-set-spmf-iface-look*:  $ys' \in \text{set-spmf } (\text{iface-look } s \ x) \longleftrightarrow$   
 $(\exists \text{ msg parties. } s = (\text{State-Store } \text{msg}, \text{ parties}) \wedge ys' = (\text{Out-Look } (\text{leak } \text{msg}),$   
 $\text{State-Store } \text{msg}, \text{ parties}))$   
 $\langle \text{proof} \rangle$

**lemma** *in-set-spmf-iface-fedit*:  $ys' \in \text{set-spmf } (\text{iface-fedit } s \ x) \longleftrightarrow$   
 $(\exists \text{ msg parties msg'. } s = (\text{State-Store } \text{msg}, \text{ parties}) \wedge x = (\text{Inp-Fedit } \text{msg}') \wedge$   
 $ys' = (\text{if } \text{editable} \text{ then } (\text{Out-Fedit}, \text{State-Collect } \text{msg}', \text{ parties}) \text{ else } (\text{Out-Fedit},$   
 $\text{State-Collect } \text{msg}, \text{ parties})))$   
 $\langle \text{proof} \rangle$

## 10.4 Defining the user interfaces

**fun** *iface-alice* :: ('msg state, 'msg iusr-alice, oustr-alice) oracle'  
**where**  
*iface-alice* (State-Void, parties) (Inp-Send msg) =  
 (if Alice  $\in$  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 oustr-bob) oracle'  
**where**  
*iface-bob* (State-Collect msg, parties) - =  
 (if Bob  $\in$  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 oustr) oracle'  
**where**  
*iface-usr*  $\equiv$  plus-oracle *iface-alice* *iface-bob*

**lemma** *in-set-spmf-iface-alice*:  $ys' \in \text{set-spmf } (\text{iface-alice } s \ x) \longleftrightarrow$   
 $(\exists \text{ parties msg. } s = (\text{State-Void}, \text{ parties}) \wedge x = \text{Inp-Send } \text{msg} \wedge \text{Alice} \in \text{parties}$   
 $\wedge ys' = (\text{Out-Send}, \text{State-Store } \text{msg}, \text{ parties}))$   
 $\langle \text{proof} \rangle$

**lemma** *in-set-spmf-iface-bob*:  $ys' \in \text{set-spmf } (\text{iface-bob } s \ x) \longleftrightarrow$   
 $(\exists \text{ msg parties. } s = (\text{State-Collect } \text{msg}, \text{ parties}) \wedge \text{Bob} \in \text{parties} \wedge ys' = (\text{Out-Recv}$   
 $\text{msg}, \text{State-Collected}, \text{ parties}))$   
 $\langle \text{proof} \rangle$

## 10.5 Defining the Fused Resource

**primcorec** *core* :: ('msg state, event, 'msg iadv, 'msg iusr, 'leak oadv, 'msg oustr)  
*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 ( $\mathcal{I}$ -full  $\oplus_{\mathcal{I}}$  ( $\mathcal{I}$ -full  $\oplus_{\mathcal{I}}$   $\mathcal{I}$ -uniform (Inp-Fedit ‘ valid-messages) UNIV))
( $\mathcal{I}$ -uniform (Inp-Send ‘ valid-messages) UNIV  $\oplus_{\mathcal{I}}$  ( $\mathcal{I}$ -uniform UNIV (Out-Recv ‘
valid-messages)))
  (pred-prod (pred-s-kernel ( $\lambda$ msg. msg  $\in$  valid-messages)) ( $\lambda$ -. 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  $\oplus_{\mathcal{I}}$  ( $\mathcal{I}$ -full  $\oplus_{\mathcal{I}}$   $\mathcal{I}$ -uniform (Inp-Fedit ‘ valid-messages) UNIV))
 $\oplus_{\mathcal{I}}$   $\mathcal{I}$ -adv-rest)  $\oplus_{\mathcal{I}}$ 
  (( $\mathcal{I}$ -uniform (Inp-Send ‘ valid-messages) UNIV  $\oplus_{\mathcal{I}}$   $\mathcal{I}$ -uniform UNIV (Out-Recv
‘ valid-messages))  $\oplus_{\mathcal{I}}$   $\mathcal{I}$ -usr-rest)  $\vdash$ res resource rest  $\surd$ 
  <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  $\Rightarrow$  'msg False +
  sec: ideal-channel  $\lambda$ - :: 'msg. carrier  $\mathcal{L}$  False +
  boolean-algebra  $\mathcal{L}$ 

```

```

for

```

```

   $\mathcal{L}$  :: ('msg, 'more) boolean-algebra-scheme (structure) +

```

```

assumes

```

```

  nempty-carrier: carrier  $\mathcal{L} \neq \{\}$  and
  finite-carrier: finite (carrier  $\mathcal{L}$ )

```

```

begin

```

## 11.1 Defining user callees

**definition** *enc-callee* :: *unit*  $\Rightarrow$  *'msg sec.iusr-alice*  
 $\Rightarrow$  (*sec.ousr-alice*  $\times$  *unit*, *key.iusr-alice* + *'msg sec.iusr-alice*, *'msg key.ousr-alice*  
+ *auth.ousr-alice*) *gpv*

**where**

*enc-callee*  $\equiv$  *stateless-callee* ( $\lambda$ *inp*. *case inp of sec.Inp-Send msg*  $\Rightarrow$   
*if msg*  $\in$  *carrier*  $\mathcal{L}$  *then*

*Pause*

(*Inl key.Inp-Alice*)

( $\lambda$ *kout*. *case projl kout of key.Out-Alice key*  $\Rightarrow$

*let cipher* = *key*  $\oplus$  *msg* *in*

*Pause* (*Inr* (*auth.Inp-Send cipher*)) ( $\lambda$ -. *Done sec.Out-Send*))

*else*

*Fail*)

**definition** *dec-callee* :: *unit*  $\Rightarrow$  *sec.iusr-bob*

$\Rightarrow$  (*'msg sec.ousr-bob*  $\times$  *unit*, *key.iusr-bob* + *auth.iusr-bob*, *'msg key.ousr-bob* +  
*'msg auth.ousr-bob*) *gpv*

**where**

*dec-callee*  $\equiv$  *stateless-callee* ( $\lambda$ -.  
*Pause*

(*Inr auth.Inp-Recv*)

( $\lambda$ *cout*. *case cout of*

*Inr* (*auth.Out-Recv cipher*)  $\Rightarrow$

*Pause*

(*Inl key.Inp-Bob*)

( $\lambda$ *kout*. *case projl kout of key.Out-Bob key*  $\Rightarrow$

*Done* (*sec.Out-Recv* (*key*  $\oplus$  *cipher*)))

| -  $\Rightarrow$  *Fail*))

## 11.2 Defining adversary converter

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

**definition** *look-callee* :: *'msg astate*  $\Rightarrow$  *sec.iadv-look*

$\Rightarrow$  (*'msg sec.oadv-look*  $\times$  *'msg astate*, *sec.iadv-look*, *'msg set sec.oadv-look*) *gpv*

**where**

*look-callee*  $\equiv$   $\lambda$ *state inp*.

*Pause*

*sec.Inp-Look*

( $\lambda$ *cout*. *case cout of*

*sec.Out-Look msg-set*  $\Rightarrow$

(*case state of*

*None*  $\Rightarrow$  *do* {

*msg*  $\leftarrow$  *lift-spmf* (*spmf-of-set* (*msg-set*));

*Done* (*auth.Out-Look msg*, *Some msg*) }

| *Some msg*  $\Rightarrow$  *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**  
 $sim \equiv$   
 $let\ look-converter = converter-of-callee\ look-callee\ None\ in$   
 $ldummy-converter\ (\lambda-. key.Out-Adversary)\ (1_C\ |=\ look-converter\ |=\ 1_C)$

### 11.3 Defining event-translator

**type-synonym**  $estate = bool \times (key.party + auth.party)$  set

**abbreviation**  $einit :: estate$

**where**  
 $einit \equiv (False, \{\})$

**definition**  $sec-party-of-key-party :: key.party \Rightarrow sec.party$

**where**  
 $sec-party-of-key-party \equiv key.case-party\ sec.Alice\ sec.Bob$

**abbreviation**  $etran-base-helper :: estate \Rightarrow key.party + auth.party \Rightarrow sec.event\ list$

**where**  
 $etran-base-helper \equiv (\lambda(s-flg, s-kap)\ item.$   
 $let\ sp-of = case-sum\ sec-party-of-key-party\ id\ in$   
 $let\ se-of = (\lambdachk\ out.\ if\ s-flg \wedge chk\ then\ [out]\ else\ [])\ in$   
 $let\ chk-alice = Inl\ key.Alice \in s-kap \wedge Inr\ auth.Alice \in s-kap\ in$   
 $let\ chk-bob = Inl\ key.Bob \in s-kap \wedge Inr\ auth.Bob \in s-kap\ in$   
 $sec.case-party$   
 $(se-of\ chk-alice\ (sec.Event-Shell\ sec.Alice))$   
 $(se-of\ chk-bob\ (sec.Event-Shell\ sec.Bob))$   
 $(sp-of\ item))$

**abbreviation**  $etran-base :: (estate, key.party + auth.party, sec.event\ list)$  oracle'

**where**  
 $etran-base \equiv (\lambda(s-flg, s-kap)\ item.$   
 $let\ s-kap' = insert\ item\ s-kap\ in$   
 $let\ event = etran-base-helper\ (s-flg, s-kap')\ item\ in$   
 $if\ item \notin s-kap\ then\ return-spmf\ (event, s-flg, s-kap')\ else\ return-pmf\ None)$

**fun**  $etran :: (estate, key.event + auth.event, sec.event\ list)$  oracle'

**where**  
 $etran\ state\ (Inl\ (key.Event-Shell\ party)) = etran-base\ state\ (Inl\ party)$   
 $| etran\ (False, s-kap)\ (Inl\ key.Event-Kernel) =$   
 $(let\ check-alice = Inl\ key.Alice \in s-kap \wedge Inr\ auth.Alice \in s-kap\ in$   
 $let\ check-bob = Inl\ key.Bob \in s-kap \wedge Inr\ auth.Bob \in s-kap\ in$   
 $let\ e-alice = if\ check-alice\ then\ [sec.Event-Shell\ sec.Alice]\ else\ []\ in$   
 $let\ e-bob = if\ check-bob\ then\ [sec.Event-Shell\ sec.Bob]\ else\ []\ in$

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

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

**lemma** *sec-party-of-key-party-simps* [simp]:  
 $\text{sec-party-of-key-party key.Alice} = \text{sec.Alice}$   
 $\text{sec-party-of-key-party key.Bob} = \text{sec.Bob}$   
 ⟨proof⟩

**lemma** *sec-party-of-key-party-eq-simps* [simp]:  
 $\text{sec-party-of-key-party } p = \text{sec.Alice} \longleftrightarrow p = \text{key.Alice}$   
 $\text{sec-party-of-key-party } p = \text{sec.Bob} \longleftrightarrow p = \text{key.Bob}$   
 ⟨proof⟩

**lemma** *key-case-party-collapse* [simp]:  $\text{key.case-party } x \ x \ p = x$   
 ⟨proof⟩

**lemma** *sec-case-party-collapse* [simp]:  $\text{sec.case-party } x \ x \ p = x$   
 ⟨proof⟩

**lemma** *Alice-in-sec-party-of-key-party* [simp]:  
 $\text{sec.Alice} \in \text{sec-party-of-key-party } 'P \longleftrightarrow \text{key.Alice} \in P$   
 ⟨proof⟩

**lemma** *Bob-in-sec-party-of-key-party* [simp]:  
 $\text{sec.Bob} \in \text{sec-party-of-key-party } 'P \longleftrightarrow \text{key.Bob} \in P$   
 ⟨proof⟩

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

## 11.4 Defining Ideal and Real constructions

**context**

**fixes**

$\text{key-rest} :: ('key\text{-s-rest}, \text{key.event}, 'key\text{-iadv-rest}, 'key\text{-iusr-rest}, 'key\text{-oadv-rest}, 'key\text{-ousr-rest}) \text{rest-wstate}$  **and**

$\text{auth-rest} :: ('auth\text{-s-rest}, \text{auth.event}, 'auth\text{-iadv-rest}, 'auth\text{-iusr-rest}, 'auth\text{-oadv-rest}, 'auth\text{-ousr-rest}) \text{rest-wstate}$

**begin**

**definition** *ideal-rest*

**where**

$\text{ideal-rest} \equiv \text{translate-rest einit etran } (\text{parallel-rest key-rest auth-rest})$

**definition** *ideal-resource*

**where**

$ideal-resource \equiv (sim \mid= 1_C) \mid= 1_C \mid= 1_C \triangleright (sec.resource \ ideal-rest)$

**definition** *real-resource*

**where**

$real-resource \equiv attach-c1f22-c1f22 (CNV \ enc-callee \ ()) (CNV \ dec-callee \ ())$   
 $(key.resource \ key-rest) (auth.resource \ auth-rest)$

## 11.5 Wiring and simplifying the Ideal construction

**definition** *ideal-s-core'* ::  $((- \times 'msg \ astate \times -) \times -) \times estate \times 'msg \ sec.state$

**where**

$ideal-s-core' \equiv ((((), \ None, \ ()), \ ()), \ (False, \ \{\}), \ sec.State-Void, \ \{\})$

**definition** *ideal-s-rest'* ::  $- \times 'key-s-rest \times 'auth-s-rest$

**where**

$ideal-s-rest' \equiv ((((), \ ()), \ rinit \ key-rest, \ rinit \ auth-rest)$

**primcorec** *ideal-core'* ::  $((unit \times - \times unit) \times unit) \times -, -, key.iadv + -, -, -, -)$   
*core*

**where**

$cpoke \ ideal-core' = (\lambda(s-advusr, \ s-event, \ s-core) \ event. \ do \ \{$   
 $(events, \ s-event') \leftarrow (etran \ s-event \ event);$   
 $s-core' \leftarrow foldl-spmf \ sec.poke \ (return-spmf \ s-core) \ events;$   
 $return-spmf \ (s-advusr, \ s-event', \ s-core')$   
 $\})$

$| \ cfunc-adv \ ideal-core' = (\lambda((s-adv, \ s-usr), \ s-core) \ iadv.$

$let \ handle-l = (\lambda-. \ Done \ (Inl \ key.Out-Adversary, \ s-adv)) \ in$

$let \ handle-r = (\lambdaqr. \ map-gpv \ (map-prod \ Inr \ id) \ id \ ((1_I \ \ddagger_I \ look-callee \ \ddagger_I \ 1_I)$

$s-adv \ qr)) \ in$

$map-spmf$

$(\lambda((oadv, \ s-adv'), \ s-core'). \ (oadv, \ (s-adv', \ s-usr), \ s-core'))$

$(exec-gpv \ \ddagger \ sec.iface-adv \ (case-sum \ handle-l \ handle-r \ iadv) \ s-core))$

$| \ cfunc-usr \ ideal-core' = \ \ddagger \ \ddagger \ sec.iface-usr$

**primcorec** *ideal-rest'* ::  $((unit \times unit) \times -, -, -, -, -, -)$  *rest-scheme*

**where**

$rinit \ ideal-rest' = ((((), \ ()), \ rinit \ key-rest, \ rinit \ auth-rest)$

$| \ rfunc-adv \ ideal-rest' = \ \ddagger \ (parallel-eoracle \ (rfunc-adv \ key-rest) \ (rfunc-adv \ auth-rest))$

$| \ rfunc-usr \ ideal-rest' = \ \ddagger \ (parallel-eoracle \ (rfunc-usr \ key-rest) \ (rfunc-usr \ auth-rest))$

### 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'* ::  $- \times 'msg \ key.state \times 'msg \ auth.state$

**where**

$real-s-core' \equiv ((((), (), ()), (key.PState-Store, \{\}), (auth.State-Void, \{\})))$

**definition**  $real-s-rest'$

**where**

$real-s-rest' \equiv ideal-s-rest'$

**primcorec**  $real-core' :: ((unit \times -) \times -, -, -, -, -, -) core$

**where**

$cpoke\ real-core' = (\lambda(s-advusr, s-core) event.$

$map-spmf (Pair\ s-advusr) (parallel-handler\ key.poke\ auth.poke\ s-core\ event))$

$| cfunc-adv\ real-core' = \dagger(key.iface-adv\ \ddagger_O\ auth.iface-adv)$

$| cfunc-usr\ real-core' = (\lambda((s-adv, s-usr), s-core)\ iusr.$

$let\ handle-req = lsumr \circ map-sum\ id\ (rsuml \circ map-sum\ swap-sum\ id \circ lsumr)$

$\circ rsuml\ in$

$let\ handle-ret = lsumr \circ (map-sum\ id\ (rsuml \circ (map-sum\ swap-sum\ id \circ lsumr))) \circ rsuml\ in$

$map-spmf$

$(\lambda((ousr, s-usr'), s-core'). (ousr, (s-adv, s-usr'), s-core'))$

$(exec-gpv$

$(key.iface-usr\ \ddagger_O\ auth.iface-usr)$

$(map-gpv'\ id\ handle-req\ handle-ret\ ((enc-callee\ \ddagger_I\ dec-callee)\ s-usr\ iusr))$

$s-core))$

**definition**  $real-rest'$

**where**

$real-rest' \equiv ideal-rest'$

### 11.6.1 The real attachment lemma

**private lemma**  $WT-callee-real1: ((\mathcal{I}\text{-full} \oplus_{\mathcal{I}} \mathcal{I}\text{-full}) \oplus_{\mathcal{I}} (\mathcal{I}\text{-full} \oplus_{\mathcal{I}} \mathcal{I}\text{-full})) \oplus_{\mathcal{I}} ((\mathcal{I}\text{-full} \oplus_{\mathcal{I}} \mathcal{I}\text{-full}) \oplus_{\mathcal{I}} (\mathcal{I}\text{-full} \oplus_{\mathcal{I}} \mathcal{I}\text{-full})) \vdash c$

$(key.fuse\ key-rest\ \ddagger_O\ auth.fuse\ auth-rest)\ s\ \checkmark$

$\langle proof \rangle$  **lemma**  $WT-callee-real2: (\mathcal{I}\text{-full} \oplus_{\mathcal{I}} \mathcal{I}\text{-full}) \oplus_{\mathcal{I}} (((\mathcal{I}\text{-full} \oplus_{\mathcal{I}} \mathcal{I}\text{-full}) \oplus_{\mathcal{I}} (\mathcal{I}\text{-full} \oplus_{\mathcal{I}} \mathcal{I}\text{-full})) \oplus_{\mathcal{I}} \mathcal{I}\text{-full}) \vdash c$

$fused-resource.fuse\ (parallel-core\ key.core\ auth.core)\ (parallel-rest\ key-rest\ auth-rest)\ s\ \checkmark$

$\langle proof \rangle$

**lemma**  $attach-real: real-resource = RES\ (fused-resource.fuse\ real-core'\ real-rest')$   
 $(real-s-core', real-s-rest')$

$\langle proof \rangle$

## 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**

$$a-I \equiv \lambda(x, y). ((((), x, ()), ()), y)$$

**private abbreviation**

$$a-R \equiv \lambda x. ((((), ()), ()), x)$$

**abbreviation**

$$\begin{aligned} asm-act &\equiv (\lambda flg \ pset-sec \ pset-key \ pset-auth \ pset-union. \\ pset-union &= pset-key <+> pset-auth \wedge \\ flg &\longrightarrow pset-sec = sec-party-of-key-party \ ' pset-key \cap \ pset-auth)) \end{aligned}$$

**private inductive**  $S :: (((- \times 'msg \ option \times -) \times -) \times estate \times 'msg \ sec.state) \Rightarrow (- \times 'msg \ key.state \times 'msg \ auth.state) \Rightarrow bool$

**where**

- $(Auth = a)@(Key = 0)$ 
  - $s-0-0: S \ (return\text{-}spmf \ (a-I \ (None, \ (False, \ s\text{-}act\text{-}ka), \ sec.State\text{-}Void, \ s\text{-}act\text{-}s)))$
  - $(return\text{-}spmf \ (a-R \ ((key.PState\text{-}Store, \ s\text{-}act\text{-}k), \ auth.State\text{-}Void, \ s\text{-}act\text{-}a)))$
  - if**  $asm\text{-}act \ False \ s\text{-}act\text{-}s \ s\text{-}act\text{-}k \ s\text{-}act\text{-}a \ s\text{-}act\text{-}ka$  **and**  $s\text{-}act\text{-}s = \{\}$
- $(Auth = a)@(Key = 1)$ 
  - |  $s-0-1: S \ (return\text{-}spmf \ (a-I \ (None, \ (True, \ s\text{-}act\text{-}ka), \ sec.State\text{-}Void, \ s\text{-}act)))$
  - $(map\text{-}spmf \ (\lambda key. \ a-R \ ((key.State\text{-}Store \ key, \ s\text{-}act\text{-}k), \ auth.State\text{-}Void, \ s\text{-}act\text{-}a)))$
  - $(spmf\text{-}of\text{-}set \ (carrier \ \mathcal{L}))$
  - if**  $asm\text{-}act \ True \ s\text{-}act \ s\text{-}act\text{-}k \ s\text{-}act\text{-}a \ s\text{-}act\text{-}ka$
- $../(Auth = a)@(Key = 1) \ # \ w1$ 
  - |  $s-1-1: S \ (return\text{-}spmf \ (a-I \ (None, \ (True, \ s\text{-}act\text{-}ka), \ sec.State\text{-}Store \ msg, \ s\text{-}act\text{-}s)))$
  - $(map\text{-}spmf \ (\lambda key. \ a-R \ ((key.State\text{-}Store \ key, \ s\text{-}act\text{-}k), \ auth.State\text{-}Store \ (key \oplus \ msg), \ s\text{-}act\text{-}a)))$
  - $(spmf\text{-}of\text{-}set \ (carrier \ \mathcal{L}))$
  - if**  $asm\text{-}act \ True \ s\text{-}act\text{-}s \ s\text{-}act\text{-}k \ s\text{-}act\text{-}a \ s\text{-}act\text{-}ka$  **and**  $key.Alice \in s\text{-}act\text{-}k$  **and**  $auth.Alice \in s\text{-}act\text{-}a$  **and**  $msg \in carrier \ \mathcal{L}$
  - |  $s-2-1: S \ (return\text{-}spmf \ (a-I \ (None, \ (True, \ s\text{-}act\text{-}ka), \ sec.State\text{-}Collect \ msg, \ s\text{-}act\text{-}s)))$
  - $(map\text{-}spmf \ (\lambda key. \ a-R \ ((key.State\text{-}Store \ key, \ s\text{-}act\text{-}k), \ auth.State\text{-}Collect \ (key \oplus \ msg), \ s\text{-}act\text{-}a)))$
  - $(spmf\text{-}of\text{-}set \ (carrier \ \mathcal{L}))$
  - if**  $asm\text{-}act \ True \ s\text{-}act\text{-}s \ s\text{-}act\text{-}k \ s\text{-}act\text{-}a \ s\text{-}act\text{-}ka$  **and**  $key.Alice \in s\text{-}act\text{-}k$  **and**  $auth.Alice \in s\text{-}act\text{-}a$  **and**  $msg \in carrier \ \mathcal{L}$
  - |  $s-3-1: S \ (return\text{-}spmf \ (a-I \ (None, \ (True, \ s\text{-}act\text{-}ka), \ sec.State\text{-}Collected, \ s\text{-}act\text{-}s)))$
  - $(map\text{-}spmf \ (\lambda key. \ a-R \ ((key.State\text{-}Store \ key, \ s\text{-}act\text{-}k), \ auth.State\text{-}Collected, \ s\text{-}act\text{-}a)))$
  - $(spmf\text{-}of\text{-}set \ (carrier \ \mathcal{L}))$
  - if**  $asm\text{-}act \ True \ s\text{-}act\text{-}s \ s\text{-}act\text{-}k \ s\text{-}act\text{-}a \ s\text{-}act\text{-}ka$  **and**  $s\text{-}act\text{-}k = \{key.Alice, \ key.Bob\}$  **and**  $s\text{-}act\text{-}a = \{auth.Alice, \ auth.Bob\}$
- $../(Auth = a)@(Key = 1) \ # \ look$ 
  - |  $s-1'-1: S \ (return\text{-}spmf \ (a-I \ (Some \ (key \oplus \ msg), \ (True, \ s\text{-}act\text{-}ka), \ sec.State\text{-}Store \ msg, \ s\text{-}act\text{-}s)))$
  - $(return\text{-}spmf \ (a-R \ ((key.State\text{-}Store \ key, \ s\text{-}act\text{-}k), \ auth.State\text{-}Store \ (key \oplus \ msg), \ s\text{-}act\text{-}a)))$
  - if**  $asm\text{-}act \ True \ s\text{-}act\text{-}s \ s\text{-}act\text{-}k \ s\text{-}act\text{-}a \ s\text{-}act\text{-}ka$  **and**  $key.Alice \in s\text{-}act\text{-}k$  **and**  $auth.Alice \in s\text{-}act\text{-}a$  **and**  $msg \in carrier \ \mathcal{L}$  **and**  $key \in carrier \ \mathcal{L}$

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

**private lemma**  $\text{trace-eq-core}$ :  $\text{trace-core-eq}$   $\text{ideal-core}'$   $\text{real-core}'$

$UNIV$  ( $UNIV$   $\langle + \rangle$   $UNIV$   $\langle + \rangle$   $UNIV$   $\langle + \rangle$  ( $\text{auth.Inp-Fedit}$  '  $\text{carrier } \mathcal{L}$ )  
 $((\text{sec.Inp-Send}$  '  $\text{carrier } \mathcal{L}$ )  $\langle + \rangle$   $UNIV$ )  
 $(\text{return-spmf}$   $\text{ideal-s-core}'$ )  $(\text{return-spmf}$   $\text{real-s-core}'$ )  
 $\langle \text{proof} \rangle$

### 11.7.2 Proving the trace equivalence of fused cores and rests

**private definition**  $\mathcal{I}\text{-adv-core}$  :: ( $\text{key.iadv}$  + '  $\text{msg}$   $\text{auth.iadv}$ ,  $\text{key.oadv}$  + '  $\text{msg}$   $\text{auth.oadv}$ )  $\mathcal{I}$

**where**  $\mathcal{I}\text{-adv-core} \equiv \mathcal{I}\text{-full} \oplus_{\mathcal{I}} (\mathcal{I}\text{-full} \oplus_{\mathcal{I}} (\mathcal{I}\text{-full} \oplus_{\mathcal{I}} \mathcal{I}\text{-uniform} (\text{sec.Inp-Fedit}$  '  $\text{carrier } \mathcal{L}$ ))  $UNIV$ ))

**private definition**  $\mathcal{I}\text{-usr-core}$  :: ('  $\text{msg}$   $\text{sec.iusr}$ , '  $\text{msg}$   $\text{sec.ousr}$ )  $\mathcal{I}$

**where**  $\mathcal{I}\text{-usr-core} \equiv \mathcal{I}\text{-uniform} (\text{sec.Inp-Send}$  ' ( $\text{carrier } \mathcal{L}$ ))  $UNIV$   $\oplus_{\mathcal{I}}$   $\mathcal{I}\text{-uniform}$   $UNIV$  ( $\text{sec.Out-Recv}$  '  $\text{carrier } \mathcal{L}$ )

**private definition**  $\text{invar-ideal}'$  :: ((-  $\times$  '  $\text{msg}$   $\text{astate}$   $\times$  -)  $\times$  -)  $\times$   $\text{estate}$   $\times$  '  $\text{msg}$   $\text{sec.state} \Rightarrow \text{bool}$

**where**  $\text{invar-ideal}' = \text{pred-prod}$  ( $\text{pred-prod}$  ( $\text{pred-prod}$  ( $\lambda\text{-}$ .  $\text{True}$ ) ( $\text{pred-prod}$  ( $\text{pred-option}$  ( $\lambda x. x \in \text{carrier } \mathcal{L}$ )) ( $\lambda\text{-}$ .  $\text{True}$ ))) ( $\lambda\text{-}$ .  $\text{True}$ )) ( $\text{pred-prod}$  ( $\lambda\text{-}$ .  $\text{True}$ ) ( $\text{pred-prod}$  ( $\text{sec.pred-s-kernel}$  ( $\lambda x. x \in \text{carrier } \mathcal{L}$ )) ( $\lambda\text{-}$ .  $\text{True}$ ))) ( $\lambda\text{-}$ .  $\text{True}$ ))

**private definition**  $\text{invar-real}'$  :: -  $\times$  ('  $\text{msg}$   $\text{key.s-kernel}$   $\times$  -)  $\times$  '  $\text{msg}$   $\text{sec.s-kernel}$   $\times$  -  $\Rightarrow \text{bool}$

**where**  $\text{invar-real}' = \text{pred-prod}$  ( $\lambda\text{-}$ .  $\text{True}$ ) ( $\text{pred-prod}$  ( $\text{pred-prod}$  ( $\text{key.pred-s-kernel}$  ( $\lambda x. x \in \text{carrier } \mathcal{L}$ )) ( $\lambda\text{-}$ .  $\text{True}$ )) ( $\text{pred-prod}$  ( $\text{sec.pred-s-kernel}$  ( $\lambda x. x \in \text{carrier } \mathcal{L}$ )) ( $\lambda\text{-}$ .  $\text{True}$ ))) ( $\lambda\text{-}$ .  $\text{True}$ ))

**lemma**  $\text{invar-ideal-s-core}'$  [ $\text{simp}$ ]:  $\text{invar-ideal}'$   $\text{ideal-s-core}'$

$\langle \text{proof} \rangle$

**lemma**  $\text{invar-real-s-core}'$  [ $\text{simp}$ ]:  $\text{invar-real}'$   $\text{real-s-core}'$

$\langle \text{proof} \rangle$

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

*<proof>*

**lemma** *WT-ideal-rest'* [*WT-intro*]:

**assumes** *WT-rest I-adv-restk I-usr-restk I-key-rest key-rest*

**and** *WT-rest I-adv-resta I-usr-resta I-auth-rest auth-rest*

**shows** *WT-rest (I-adv-restk  $\oplus_{\mathcal{I}}$  I-adv-resta) (I-usr-restk  $\oplus_{\mathcal{I}}$  I-usr-resta) ( $\lambda(-, s-rest)$ . pred-prod I-key-rest I-auth-rest s-rest) ideal-rest'*

*<proof>*

**lemma** *WT-real-core'* [*WT-intro*]: *WT-core I-adv-core I-usr-core invar-real' real-core'*  
*<proof>* **lemma** *trace-eq-sec*:

**fixes** *I-adv-restk I-adv-resta I-usr-restk I-usr-resta*

**defines** *outs-adv  $\equiv$  (UNIV  $\langle + \rangle$  UNIV  $\langle + \rangle$  UNIV  $\langle + \rangle$  sec.Inp-Fedit 'carrier  $\mathcal{L}$ )  $\langle + \rangle$  outs-I (I-adv-restk  $\oplus_{\mathcal{I}}$  I-adv-resta)*

**and** *outs-usr  $\equiv$  (sec.Inp-Send 'carrier  $\mathcal{L}$   $\langle + \rangle$  UNIV)  $\langle + \rangle$  outs-I (I-usr-restk  $\oplus_{\mathcal{I}}$  I-usr-resta)*

**assumes** *WT-key* [*WT-intro*]: *WT-rest I-adv-restk I-usr-restk I-key-rest key-rest*

**and** *WT-auth* [*WT-intro*]: *WT-rest I-adv-resta I-usr-resta I-auth-rest auth-rest*

**shows** *(outs-adv  $\langle + \rangle$  outs-usr)  $\vdash_C$  fused-resource.fuse ideal-core' ideal-rest' ((ideal-s-core', ideal-s-rest'))  $\approx$*

*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 I-adv-core-def I-usr-core-def*]:

**fixes** *I-adv-restk I-adv-resta I-usr-restk I-usr-resta*

**defines**  *$\mathcal{I} \equiv$  (I-adv-core  $\oplus_{\mathcal{I}}$  (I-adv-restk  $\oplus_{\mathcal{I}}$  I-adv-resta))  $\oplus_{\mathcal{I}}$  (I-usr-core  $\oplus_{\mathcal{I}}$  (I-usr-restk  $\oplus_{\mathcal{I}}$  I-usr-resta))*

**assumes** [*WT-intro*]: *WT-rest I-adv-restk I-usr-restk I-key-rest key-rest*

**and** [*WT-intro*]: *WT-rest I-adv-resta I-usr-resta I-auth-rest auth-rest*

**and** *exception-I  $\mathcal{I} \vdash_g D \checkmark$*

**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* ()  
 $\checkmark$   
 ⟨*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* ()  
 $\checkmark$   
 ⟨*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*  $\equiv$  (( $\mathcal{I}$ -*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*  $\equiv$   $\mathcal{I}$ -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*  $\equiv$   $\mathcal{I}$ -*usr-restk*  $\oplus_{\mathcal{I}}$   $\mathcal{I}$ -*usr-resta*

**and**  $\mathcal{I}$ -*ideal*  $\equiv$  ( $\mathcal{I}$ -*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*  $\models_{1C}$ )  $\mathcal{I}$ -*real*  $\mathcal{I}$ -*ideal* ( $\mathcal{I}$ -*common-core*  $\oplus_{\mathcal{I}}$   $\mathcal{I}$ -*common-rest*)  $\mathcal{A}$  0

$\langle proof \rangle$

**end**

**end**

## 11.9 Asymptotic security

**locale** *one-time-pad'* =

**fixes**  $\mathcal{L} :: security \Rightarrow ('msg, 'more) boolean-algebra-scheme$

**assumes** *one-time-pad* [*locale-witness*]:  $\bigwedge \eta. one-time-pad (\mathcal{L} \eta)$

**begin**

**sublocale** *one-time-pad*  $\mathcal{L} \eta$  **for**  $\eta$   $\langle proof \rangle$

**definition** *real-resource'* **where** *real-resource'* *rest1 rest2*  $\eta = real-resource\ TYPE(-)$   
 $TYPE(-) \eta (rest1 \eta) (rest2 \eta)$

**definition** *ideal-resource'* **where** *ideal-resource'* *rest1 rest2*  $\eta = sec.resource \eta$   
 $(ideal-rest (rest1 \eta) (rest2 \eta))$

**definition** *sim'* **where** *sim'*  $\eta = (sim \mid= 1_C)$

**context**

**fixes**

*key-rest* ::  $nat \Rightarrow ('key-s-rest, key.event, 'key-iadv-rest, 'key-iusr-rest, 'key-oadv-rest,$   
 $'key-ousr-rest) rest-wstate$  **and**

*auth-rest* ::  $nat \Rightarrow ('auth-s-rest, auth.event, 'auth-iadv-rest, 'auth-iusr-rest,$   
 $'auth-oadv-rest, 'auth-ousr-rest) rest-wstate$  **and**

*I-adv-restk* **and** *I-adv-resta* **and** *I-usr-restk* **and** *I-usr-resta* **and** *I-key-rest*  
**and** *I-auth-rest*

**assumes**

*WT-key-res*:  $\bigwedge \eta. WT-rest (I-adv-restk \eta) (I-usr-restk \eta) (I-key-rest \eta) (key-rest$   
 $\eta)$  **and**

*WT-auth-rest*:  $\bigwedge \eta. WT-rest (I-adv-resta \eta) (I-usr-resta \eta) (I-auth-rest \eta)$   
 $(auth-rest \eta)$

**begin**

**theorem** *secure'*:

**defines** *I-real*  $\equiv \lambda \eta. ((I-full \oplus_{\mathcal{I}} (I-full \oplus_{\mathcal{I}} (I-full \oplus_{\mathcal{I}} I-uniform (sec.Inp-Fedit$   
 $'(carrier (\mathcal{L} \eta))) UNIV))) \oplus_{\mathcal{I}} (I-adv-restk \eta \oplus_{\mathcal{I}} I-adv-resta \eta))$

**and** *I-common*  $\equiv \lambda \eta. ((I-uniform (sec.Inp-Send '(carrier (\mathcal{L} \eta))) UNIV \oplus_{\mathcal{I}}$   
 $I-uniform UNIV (sec.Out-Recv '(carrier (\mathcal{L} \eta))) \oplus_{\mathcal{I}} (I-usr-restk \eta \oplus_{\mathcal{I}} I-usr-resta$   
 $\eta))$

**and** *I-ideal*  $\equiv \lambda \eta. (I-full \oplus_{\mathcal{I}} (I-full \oplus_{\mathcal{I}} I-uniform (sec.Inp-Fedit '(carrier$   
 $(\mathcal{L} \eta))) UNIV)) \oplus_{\mathcal{I}} (I-adv-restk \eta \oplus_{\mathcal{I}} I-adv-resta \eta)$

**shows** *constructive-security-obsf'* (*real-resource'* *key-rest auth-rest*) (*ideal-resource'*  
*key-rest auth-rest*) *sim'* *I-real* *I-ideal* *I-common* *A*

$\langle proof \rangle$

**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 <⊕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 =  $\mathbf{g}$  [^] x;
      Pause
      (Inl (auth.Inp-Send msg))
      (λrsp. 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  $\mathcal{G}$ ));
    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**

$cpoke\ ideal-core-alt = cpoke\ (translate-core\ etran\ key.core)$   
 $| cfunc-adv\ ideal-core-alt = \dagger(cfunc-adv\ key.core) \oplus_O (\lambda(se, sc)\ q.\ do\ \{$   
 $\quad ((out, es), se') \leftarrow leak\ se\ q;$   
 $\quad sc' \leftarrow foldl-spmf\ (cpoke\ key.core)\ (return-spmf\ sc)\ es;$   
 $\quad return-spmf\ (out, se', sc')\ \})$   
 $| cfunc-usr\ ideal-core-alt = cfunc-usr\ (translate-core\ etran\ key.core)$

**primcorec** *ideal-rest-alt*

**where**

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

**primcorec** *ideal-rest*

**where**

$rinit\ ideal-rest = (einit, rinit\ ideal-rest-alt)$   
 $| rfunc-adv\ ideal-rest = (\lambda s\ q.\ case\ q\ of$   
 $\quad Inl\ ql \Rightarrow map-spmf\ (apfst\ (map-prod\ Inl\ id))\ (leak^\dagger\ s\ ql)$   
 $\quad | Inr\ qr \Rightarrow map-spmf\ (apfst\ (map-prod\ Inr\ id))\ (translate-eoracle\ etran\ \dagger(rfunc-adv\ ideal-rest-alt)\ s\ qr))$   
 $| rfunc-usr\ ideal-rest = translate-eoracle\ etran\ \dagger(rfunc-usr\ ideal-rest-alt)$

**definition** *ideal-resource*

**where**

$ideal-resource \equiv$   
 $(let\ sim = CNV\ sim-callee\ None\ in$   
 $\quad attach\ ((sim\ |=\ 1_C) \odot\ lassocr_C\ |=\ 1_C\ |=\ 1_C)\ (key.resource\ ideal-rest))$

**definition** *real-resource*

**where**

$real-resource \equiv$   
 $(let\ dh-wiring = parallel-wiring \odot (CNV\ alice-callee\ CState-Void\ |=\ CNV\ bob-callee\ CState-Void) \odot parallel-wiring \odot (1_C\ |=\ swap_C)\ in$

$attach \ ((1_C \models 1_C) \models rasso_{1_C} \odot (dh-wiring \models 1_C)) \odot fused-wiring)$   
 $((auth.resource \ auth1-rest) \parallel (auth.resource \ auth2-rest))$

## 12.5 Wiring and simplifying the Ideal construction

**abbreviation** *basic-rest-sinit*

**where**

$basic-rest-sinit \equiv (((), ()), rinit \ auth1-rest, rinit \ auth2-rest)$

**primcorec** *basic-rest* ::  $((unit \times unit) \times -, -, -, -, -, -, -)$  *rest-scheme*

**where**

$rinit \ basic-rest = (rinit \ auth1-rest, rinit \ auth2-rest)$

|  $rfunc-adv \ basic-rest = \dagger(parallel-eoracle \ (rfunc-adv \ auth1-rest) \ (rfunc-adv \ auth2-rest))$

|  $rfunc-usr \ basic-rest = \dagger(parallel-eoracle \ (rfunc-usr \ auth1-rest) \ (rfunc-usr \ auth2-rest))$

**definition** *ideal-s-core'* ::  $('grp \ astate \times -) \times - \times 'grp \ key.state$

**where**

$ideal-s-core' \equiv ((None, ()), einit, key.PState-Store, \{\})$

**definition** *ideal-s-rest'*

**where**

$ideal-s-rest' \equiv basic-rest-sinit$

**primcorec** *ideal-core'*

**where**

$cpoke \ ideal-core' = (\lambda(s-cnv, s-event, s-core) \ event. \ do \ \{$

$(events, s-event') \leftarrow (etran \ s-event \ event);$

$s-core' \leftarrow foldl-spmf \ key.poke \ (return-spmf \ s-core) \ events;$

$return-spmf \ (s-cnv, s-event', s-core') \ \}$

|  $cfunc-adv \ ideal-core' = (\lambda((s-sim, -), s-event-core) \ q.$

$map-spmf$

$(\lambda((out, s-sim'), s-event-core'). (out, (s-sim', ()), s-event-core'))$

$(exec-gpv$

$(\dagger key.iface-adv \oplus_O \ (\lambda(se, sc) \ isim. \ do \ \{$

$((out, es), se') \leftarrow leak \ se \ isim;$

$sc' \leftarrow foldl-spmf \ (cpoke \ key.core) \ (return-spmf \ sc) \ es;$

$return-spmf \ (out, se', sc') \ \}$

$(sim-callee \ s-sim \ q) \ s-event-core))$

|  $cfunc-usr \ ideal-core' = (\lambda(s-cnv, s-core) \ q.$

$map-spmf \ (\lambda(out, s-core'). (out, s-cnv, s-core')) \ (\dagger key.iface-usr \ s-core \ q))$

**primcorec** *ideal-rest'*

**where**

$rinit \ ideal-rest' = rinit \ basic-rest$

|  $rfunc-adv \ ideal-rest' = (\lambda s \ q. \ map-spmf \ (apfst \ (apsnd \ (map \ Inr))) \ (rfunc-adv \ basic-rest \ s \ q))$

|  $rfunc-usr \ ideal-rest' = ($

$let \ handle = map-sum \ (\lambda- :: icnv-alice. \ Out-Activation-Alice) \ (\lambda- :: icnv-bob.$

$Out-Activation-Bob) \ in$

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

### 12.5.1 The ideal attachment lemma

**context**

**begin**

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

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

**lemma** *attach-ideal*:  $\text{ideal-resource} = \text{RES } (\text{fused-resource.fuse ideal-core}' \text{ ideal-rest}'$   
 $(\text{ideal-s-core}', \text{ideal-s-rest}'))$   
 $\langle \text{proof} \rangle$

**end**

## 12.6 Wiring and simplifying the Real construction

**definition** *real-s-core'* ::  $(- \times 'grp \text{ cstate} \times 'grp \text{ cstate}) \times 'grp \text{ auth.state} \times 'grp \text{ auth.state}$

**where**

$\text{real-s-core}' \equiv ((((), \text{CState-Void}, \text{CState-Void}), (\text{auth.State-Void}, \{\}), (\text{auth.State-Void}, \{\})))$

**definition** *real-s-rest'*

**where**

$\text{real-s-rest}' \equiv \text{basic-rest-sinit}$

**primcorec** *real-core'* ::  $((\text{unit} \times -) \times -, -, -, -, -, -) \text{ core}$

**where**

$\text{cpoke real-core}' = (\lambda (s\text{-advusr}, s\text{-core}) \text{ event.}$   
 $\text{map-spmf } (\text{Pair } s\text{-advusr}) (\text{parallel-handler } \text{auth.poke } \text{auth.poke } s\text{-core } \text{event}))$   
 $\mid \text{cfunc-adv } \text{real-core}' = \dagger(\text{auth.iface-adv } \dagger_O \text{auth.iface-adv})$   
 $\mid \text{cfunc-usr } \text{real-core}' = (\lambda((s\text{-adv}, s\text{-usr}), s\text{-core}) \text{ iusr.}$   
 $\text{let } \text{handle-req} = \text{lsumr} \circ \text{map-sum id } (\text{rsuml} \circ \text{map-sum swap-sum id} \circ \text{lsumr})$   
 $\circ \text{rsuml} \text{ in}$   
 $\text{let } \text{handle-ret} = \text{lsumr} \circ (\text{map-sum id } (\text{rsuml} \circ (\text{map-sum swap-sum id} \circ$   
 $\text{lsumr})) \circ \text{rsuml}) \circ \text{map-sum id swap-sum in}$   
 $\text{let } \text{handle-inp} = \text{map-sum id swap-sum} \circ (\text{lsumr} \circ \text{map-sum id } (\text{rsuml} \circ$   
 $\text{map-sum swap-sum id} \circ \text{lsumr}) \circ \text{rsuml}) \text{ in}$   
 $\text{let } \text{handle-out} = \text{apfst } (\text{lsumr} \circ (\text{map-sum id } (\text{rsuml} \circ (\text{map-sum swap-sum id}$   
 $\circ \text{lsumr})) \circ \text{rsuml})) \text{ in}$

```

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* =  $1_C \mid =$  *rassocl<sub>C</sub>* ▷ *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-cnvt2*), (*auth.State-Void, parties1*),  
*s-auth2*) - =  
 (if *auth.Alice* ∈ *parties1*  
 then *return-spmf* ((*,*), (*s-key, CState-Half 0, s-cnvt2*), (*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 ⊕<sub>O</sub> (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*  $D \equiv \text{DH-adversary-curry } (\lambda xyz.$   
 $\text{run-gpv } (\text{obsf-oracle } (\text{obsf-oracle } (\lambda(tpl, s) q. \text{map-spmf } (\text{apsnd } (\text{Pair } tpl) \circ$   
 $\text{fst}) (\text{exec-gpv } (\lambda-. \text{return-spmf } (tpl, ())) (\text{interceptor } s q) ())))))$   
 $(\text{obsf-distinguish } D) (\text{OK } (\text{OK } (xyz, \text{basic-core-sinit}, \text{basic-rest-sinit}))))$

**context**

**begin**

**private abbreviation** *S-ic-asm s-cnv1 s-cnv2 s-krn1 s-krn2*  $\equiv$

$\text{let } s\text{-cnvs} = \{CState\text{-Void}\} \cup \{CState\text{-Half } 0\} \cup \{CState\text{-Full } (0, \mathbf{1})\}$  *in*  
 $\text{let } s\text{-krns} = \{\text{auth.State-Void}\} \cup \{\text{auth.State-Store } \mathbf{1}\} \cup \{\text{auth.State-Collect } \mathbf{1}\}$   
 $\cup \{\text{auth.State-Collected}\}$  *in*  
 $s\text{-cnv1} \in s\text{-cnvs} \wedge s\text{-cnv2} \in s\text{-cnvs} \wedge s\text{-krn1} \in s\text{-krns} \wedge s\text{-krn2} \in s\text{-krns}$

**private inductive** *S-ic*  $:: ('grp \times 'grp \times 'grp) \text{ spmf} \Rightarrow ('grp \text{ bc-state} \times (\text{unit} \times$   
 $\text{unit}) \times 'auth1\text{-s-rest} \times 'auth2\text{-s-rest}) \text{ spmf} \Rightarrow$   
 $(('grp \times 'grp \times 'grp) \times 'grp \text{ bc-state} \times (\text{unit} \times \text{unit}) \times 'auth1\text{-s-rest} \times$   
 $'auth2\text{-s-rest}) \text{ spmf} \Rightarrow \text{bool}$

**for**  $\mathcal{S} :: ('grp \times 'grp \times 'grp) \text{ spmf}$  **where**

*S-ic*  $\mathcal{S}$   $(\text{return-spmf } (((\text{None}, s\text{-cnv1}, s\text{-cnv2}), (s\text{-krn1}, s\text{-act1}), s\text{-krn2}, s\text{-act2}),$   
 $((), ()), s\text{-rest1}, s\text{-rest2}))$   
 $(\text{map-spmf } (\lambda x. (x, (((\text{None}, s\text{-cnv1}, s\text{-cnv2}), (s\text{-krn1}, s\text{-act1}), s\text{-krn2}, s\text{-act2}),$   
 $((), ()), s\text{-rest1}, s\text{-rest2})))) \mathcal{S}$   
**if** *S-ic-asm s-cnv1 s-cnv2 s-krn1 s-krn2*  
 $|$  *S-ic*  $\mathcal{S}$   $(\text{return-spmf } (((\text{Some } x, s\text{-cnv1}, s\text{-cnv2}), (s\text{-krn1}, s\text{-act1}), s\text{-krn2}, s\text{-act2}),$   
 $((), ()), s\text{-rest1}, s\text{-rest2}))$   
 $(\text{return-spmf } (x, (((\text{Some } x, s\text{-cnv1}, s\text{-cnv2}), (s\text{-krn1}, s\text{-act1}), s\text{-krn2}, s\text{-act2}),$   
 $((), ()), s\text{-rest1}, s\text{-rest2}))))$   
**if** *S-ic-asm s-cnv1 s-cnv2 s-krn1 s-krn2*

**private lemma** *trace-eq-intercept*:

**defines** *outs-adv*  $\equiv ((UNIV \langle + \rangle UNIV \langle + \rangle UNIV) \langle + \rangle UNIV \langle + \rangle UNIV$   
 $\langle + \rangle UNIV) \langle + \rangle UNIV \langle + \rangle UNIV$   
**and** *outs-usr*  $\equiv (UNIV \langle + \rangle UNIV) \langle + \rangle (UNIV \langle + \rangle UNIV) \langle + \rangle UNIV$   
 $\langle + \rangle UNIV$

**assumes** *lossless-spmf sample-triple*

**shows** *trace-callee-eq (fused-resource.fuse (lazy-core sample-triple) lazy-rest)*  
 $(\lambda(tpl, s) q. \text{map-spmf } (\text{apsnd } (\text{Pair } tpl) \circ \text{fst}) (\text{exec-gpv } (\lambda-. \text{return-spmf } (tpl,$   
 $())) (\text{interceptor } s q) ()))$   
 $(\text{outs-adv} \langle + \rangle \text{outs-usr})$   
 $(\text{return-spmf } (\text{basic-core-sinit}, \text{basic-rest-sinit})) (\text{pair-spmf } \text{sample-triple } (\text{return-spmf}$   
 $(\text{basic-core-sinit}, \text{basic-rest-sinit})))$   
**(is** *trace-callee-eq ?L ?R ?OI ?sl ?sr*)

*(proof)* **abbreviation** *dummy x*  $\equiv \text{TRY } \text{map-spmf } \text{OK } x \text{ ELSE } \text{return-spmf } \text{Fault}$

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

(*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*  $\equiv$  *sample-uniform* (order  $\mathcal{G}$ )

**private abbreviation** *isample-key*  $\equiv$  *spmf-of-set* (carrier  $\mathcal{G}$ )

**private abbreviation** *isample-pair-nn*  $\equiv$  *pair-spmf* *isample-nat* *isample-nat*

**private abbreviation** *isample-pair-nk*  $\equiv$  *pair-spmf* *isample-nat* *isample-key*

**private inductive** *S-il* :: (('grp *astate*  $\times$  *unit*)  $\times$  *estate*  $\times$  'grp *key.state*) *spmf*  $\Rightarrow$  'grp *bc-state* *spmf*  $\Rightarrow$  *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*  $\in$  *s-act1*

| *sil-2-0*: *S-il* (*map-spmf* ( $\lambda k.$  ((*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*  $\in$  *s-act1*

— ../(*Auth1* =*a*)@(Auth2 =0) # *look*

| *sil-1'-0*: *S-il* (*map-spmf* ( $\lambda y.$  ((*Some* (*g* [ $\Uparrow$ ] *x*, *g* [ $\Uparrow$ ] *y*), ()), (*False*, (*EState-Store*, *s-act1*), *EState-Void*, *s-act2*), *key.PState-Store*, {})) *isample-nat*)

(*map-spmf* ( $\lambda yz.$  ((*Some* (*g* [ $\Uparrow$ ] *snd* *yz*, *g* [ $\Uparrow$ ] (*x* :: *nat*), *g* [ $\Uparrow$ ] *fst* *yz*), *CState-Half 0*, *CState-Void*), (*auth.State-Store 1*, *s-act1*), *auth.State-Void*, *s-act2*)) *isample-pair-nn*)

if *auth.Alice*  $\in$  *s-act1*

| *sil-2'-0*: *S-il* (*map-spmf* ( $\lambda yk.$  ((*Some* (*g* [ $\Uparrow$ ] *x*, *g* [ $\Uparrow$ ] *fst* *yk*), ()), (*True*, (*EState-Collect*, *s-act1*), *EState-Void*, *s-act2*), *key.State-Store* (*snd* *yk*), {})) *isample-pair-nk*)

(*map-spmf* ( $\lambda yz.$  ((*Some* (*g* [ $\Uparrow$ ] *snd* *yz*, *g* [ $\Uparrow$ ] (*x* :: *nat*), *g* [ $\Uparrow$ ] *fst* *yz*), *CState-Half 0*, *CState-Void*), (*auth.State-Collect 1*, *s-act1*), *auth.State-Void*, *s-act2*)) *isample-pair-nn*)

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

*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*  $\in$  *s-act2*  
 — *../(Auth1 =a)@(Auth2 =2) # wl*  
 | *sil-1-2: S-il (map-spmf ( $\lambda k. ((None, ()), (True, (EState-Store, s-act1), EState-Collect, s-act2), key.State-Store k, s-actk)) isample-key$* )  
 (return-spmf ((None, CState-Half 0, CState-Half 0), (auth.State-Store **1**,  
 s-act1), auth.State-Collect **1**, s-act2))  
**if** *auth.Alice*  $\in$  *s-act1* **and** *auth.Alice*  $\in$  *s-act2* **and** *auth.Bob*  $\in$  *s-act2*  $\longleftrightarrow$   
*key.Alice*  $\in$  *s-actk* **and** *key.Bob*  $\notin$  *s-actk*  
 | *sil-2-2: S-il (map-spmf ( $\lambda k. ((None, ()), (True, (EState-Collect, s-act1), EState-Collect, s-act2), key.State-Store k, s-actk)) isample-key$* )  
 (return-spmf ((None, CState-Half 0, CState-Half 0), (auth.State-Collect **1**,  
 s-act1), auth.State-Collect **1**, s-act2))  
**if** *auth.Alice*  $\in$  *s-act1* **and** *auth.Alice*  $\in$  *s-act2* **and** *auth.Bob*  $\in$  *s-act2*  $\longleftrightarrow$   
*key.Alice*  $\in$  *s-actk* **and** *auth.Bob*  $\in$  *s-act1*  $\longleftrightarrow$  *key.Bob*  $\in$  *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)) isample-key*  
 (map-spmf ( $\lambda xy. ((Some (g [\ ] (z :: nat), g [\ ] fst xy, g [\ ] snd xy), CState-Half 0,$   
 CState-Full (0, **1**)), (auth.State-Collected, s-act1), auth.State-Collect **1**, s-act2))  
 isample-pair-nn)  
**if** *auth.Alice*  $\in$  *s-act1* **and** *auth.Alice*  $\in$  *s-act2* **and** *auth.Bob*  $\in$  *s-act2*  $\longleftrightarrow$   
*key.Alice*  $\in$  *s-actk* **and** *auth.Bob*  $\in$  *s-act1* **and** *key.Bob*  $\in$  *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*  $\in$  *s-act1* **and** *auth.Alice*  $\in$  *s-act2* **and** *auth.Bob*  $\in$  *s-act2*  $\longleftrightarrow$   
*key.Alice*  $\in$  *s-actk* **and** *key.Bob*  $\notin$  *s-actk* **and**  $k = g [\ ] z$  **and**  $z \in set-spmf$   
*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*  $\in$  *s-act1* **and** *auth.Alice*  $\in$  *s-act2* **and** *auth.Bob*  $\in$  *s-act2*  $\longleftrightarrow$   
*key.Alice*  $\in$  *s-actk* **and** *auth.Bob*  $\in$  *s-act1*  $\longleftrightarrow$  *key.Bob*  $\in$  *s-actk* **and**  $k = g [\ ] z$   
**and**  $z \in set-spmf isample-nat$   
 | *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*  $\in$  *s-act1* **and** *auth.Alice*  $\in$  *s-act2* **and** *auth.Bob*  $\in$  *s-act2*  $\longleftrightarrow$   
*key.Alice*  $\in$  *s-actk* **and** *auth.Bob*  $\in$  *s-act1* **and** *key.Bob*  $\in$  *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,*

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

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

**private lemma** *trac-eq-core-il*: *trace-core-eq ideal-core'* (*lazy-core DH1-sample*)  
 ((*UNIV*  $\langle + \rangle$  *UNIV*)  $\langle + \rangle$  *UNIV*  $\langle + \rangle$  *UNIV*) ((*UNIV*  $\langle + \rangle$  *UNIV*  $\langle + \rangle$   
*UNIV*)  $\langle + \rangle$  *UNIV*  $\langle + \rangle$  *UNIV*  $\langle + \rangle$  *UNIV*) (*UNIV*  $\langle + \rangle$  *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*  $\equiv$  *sample-uniform* (*order* *G*)  
**private abbreviation** *rsample-pair-nn*  $\equiv$  *pair-spmf* *rsample-nat* *rsample-nat*

**private inductive** *S-rl* :: ((*unit*  $\times$  '*grp cstate*  $\times$  '*grp cstate*)  $\times$  '*grp auth.state*  $\times$   
 '*grp auth.state*) *spmf*  
 $\Rightarrow$  (('(*grp st-state*  $\times$  '*grp cstate*  $\times$  '*grp cstate*)  $\times$  '*grp auth.state*  $\times$  '*grp auth.state*)  
*spmf*  $\Rightarrow$  *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* ( $\lambda x. ((((), \text{CState-Half } x, \text{CState-Void}), (\text{auth.State-Store}$   
     ( $\mathbf{g} [\checkmark] x), \text{s-act1}), \text{auth.State-Void}, \text{s-act2})) \text{rsample-nat}$ )  
     (*return-spmf* ((*None*, *CState-Half* 0, *CState-Void*), (*auth.State-Store*  $\mathbf{1}$ , *s-act1*),  
     *auth.State-Void*, *s-act2*)))  
   | *srl-2-0*: *S-rl* (*map-spmf* ( $\lambda x. ((((), \text{CState-Half } x, \text{CState-Void}), (\text{auth.State-Collect}$   
     ( $\mathbf{g} [\checkmark] x), \text{s-act1}), \text{auth.State-Void}, \text{s-act2})) \text{rsample-nat}$ )  
     (*return-spmf* ((*None*, *CState-Half* 0, *CState-Void*), (*auth.State-Collect*  $\mathbf{1}$ ,  
     *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** [  $\checkmark$  ] *x*), *s-act1*), *auth.State-Void*, *s-act2*))  
     (*map-spmf* ( $\lambda y$ . ((*Some* ((**g** [  $\checkmark$  ] *x*) [  $\checkmark$  ] *y*, **g** [  $\checkmark$  ] *x*, **g** [  $\checkmark$  ] *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** [  $\checkmark$  ] *x*), *s-act1*), *auth.State-Void*, *s-act2*))  
     (*map-spmf* ( $\lambda y$ . ((*Some* ((**g** [  $\checkmark$  ] *x*) [  $\checkmark$  ] *y*, **g** [  $\checkmark$  ] *x*, **g** [  $\checkmark$  ] *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* ( $\lambda y$ . (((), *CState-Void*, *CState-Half* *y*), (*auth.State-Void*, *s-act1*), *auth.State-Store* (**g** [  $\checkmark$  ] *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* ( $\lambda yx$ . (((), *CState-Half* (*snd* *yx*), *CState-Half* (*fst* *yx*)), (*auth.State-Store* (**g** [  $\checkmark$  ] *snd* *yx*), *s-act1*), *auth.State-Store* (**g** [  $\checkmark$  ] *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* ( $\lambda yx$ . (((), *CState-Half* (*snd* *yx*), *CState-Half* (*fst* *yx*)), (*auth.State-Collect* (**g** [  $\checkmark$  ] *snd* *yx*), *s-act1*), *auth.State-Store* (**g** [  $\checkmark$  ] *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** [  $\checkmark$  ] *x*), *s-act1*), *auth.State-Store* (**g** [  $\checkmark$  ] *y*), *s-act2*))  
     (*return-spmf* ((*Some* ((**g** [  $\checkmark$  ] *x*) [  $\checkmark$  ] *y*, **g** [  $\checkmark$  ] *x*, **g** [  $\checkmark$  ] *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** [  $\checkmark$  ] *x*), *s-act1*), *auth.State-Store* (**g** [  $\checkmark$  ] *y*), *s-act2*))  
     (*return-spmf* ((*Some* ((**g** [  $\checkmark$  ] *x*) [  $\checkmark$  ] *y*, **g** [  $\checkmark$  ] *x*, **g** [  $\checkmark$  ] *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** [  $\checkmark$  ] *y*), *s-act2*))  
     (*return-spmf* ((*Some* (*z*, **g** [  $\checkmark$  ] *x*, **g** [  $\checkmark$  ] *y*), *CState-Half* 0, *CState-Full* (0, **1**)), (*auth.State-Collected*, *s-act1*), *auth.State-Store* **1**, *s-act2*))  
     **if** *z* = (**g** [  $\checkmark$  ] *x*) [  $\checkmark$  ] *y*  
 — (Auth1 =a)@(Auth2 =2)  
 | *srl-0-2*: *S-rl* (*map-spmf* ( $\lambda y$ . (((), *CState-Void*, *CState-Half* *y*), (*auth.State-Void*, *s-act1*), *auth.State-Collect* (**g** [  $\checkmark$  ] *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* ( $\lambda yx$ . (((), *CState-Half* (*snd* *yx*), *CState-Half* (*fst* *yx*)), (*auth.State-Store* (**g** [  $\checkmark$  ] *snd* *yx*), *s-act1*), *auth.State-Collect* (**g** [  $\checkmark$  ] *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* ( $\lambda yx. (((), CState\text{-}Half\ (snd\ yx), CState\text{-}Half\ (fst\ yx)),$   
*(auth.State-Collect* ( $\mathbf{g}\ [\ ]\ snd\ yx$ ), *s-act1*), *auth.State-Collect* ( $\mathbf{g}\ [\ ]\ fst\ yx$ ), *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* ( $(((), CState\text{-}Half\ x, CState\text{-}Half\ y),$  (*auth.State-Store*  
 $\mathbf{g}\ [\ ]\ x$ ), *s-act1*), *auth.State-Collect* ( $\mathbf{g}\ [\ ]\ y$ ), *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* ( $(((), CState\text{-}Half\ x, CState\text{-}Half\ y),$  (*auth.State-Collect*  
 $\mathbf{g}\ [\ ]\ x$ ), *s-act1*), *auth.State-Collect* ( $\mathbf{g}\ [\ ]\ y$ ), *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* ( $(((), CState\text{-}Half\ x, CState\text{-}Full\ (y, z)),$  (*auth.State-Collected,*  
*s-act1*), *auth.State-Collect* ( $\mathbf{g}\ [\ ]\ y$ ), *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**  $z = (\mathbf{g}\ [\ ]\ x)\ [\ ]\ y$   
 — *(Auth1 =a)@(Auth2 =3)*  
 | *srl-1c-3c*: *S-rl* (*return-spmf* ( $(((), CState\text{-}Full\ (x, z), CState\text{-}Half\ y),$  (*auth.State-Store*  
 $\mathbf{g}\ [\ ]\ x$ ), *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**  $z = (\mathbf{g}\ [\ ]\ y)\ [\ ]\ x$   
 | *srl-2c-3c*: *S-rl* (*return-spmf* ( $(((), CState\text{-}Full\ (x, z), CState\text{-}Half\ y),$  (*auth.State-Collect*  
 $\mathbf{g}\ [\ ]\ x$ ), *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**  $z = (\mathbf{g}\ [\ ]\ y)\ [\ ]\ x$   
 | *srl-3c-3c*: *S-rl* (*return-spmf* ( $(((), CState\text{-}Full\ (x, z), CState\text{-}Full\ (y, z)),$  (*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**  $z = (\mathbf{g}\ [\ ]\ y)\ [\ ]\ x$   
 — *(Auth1 =0)@(Auth2 =1')*  
 | *srl-0-1'*: *S-rl* (*return-spmf* ( $(((), CState\text{-}Void, CState\text{-}Half\ y),$  (*auth.State-Void,*  
*s-act1*), *auth.State-Store* ( $\mathbf{g}\ [\ ]\ y$ ), *s-act2*))  
 (*map-spmf* ( $\lambda x. ((Some ((\mathbf{g}\ [\ ]\ x)\ [\ ]\ y, \mathbf{g}\ [\ ]\ x, \mathbf{g}\ [\ ]\ y), CState\text{-}Void,$   
*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* ( $(((), CState\text{-}Void, CState\text{-}Half\ y),$  (*auth.State-Void,*  
*s-act1*), *auth.State-Collect* ( $\mathbf{g}\ [\ ]\ y$ ), *s-act2*))  
 (*map-spmf* ( $\lambda x. ((Some ((\mathbf{g}\ [\ ]\ x)\ [\ ]\ y, \mathbf{g}\ [\ ]\ x, \mathbf{g}\ [\ ]\ y), CState\text{-}Void,$   
*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)*

$UNIV <+> UNIV) ((UNIV <+> UNIV) <+> UNIV <+> UNIV)$   
 (return-spmf real-s-core') (return-spmf basic-core-sinit)  
 <proof>

**lemma** trace-eq-fuse-rl:  $UNIV \vdash_R 1_C \mid = \text{rassocl}_C \triangleright RES$  (fused-resource.fuse  
 real-core' real-rest') (real-s-core', real-s-rest')  
 $\approx RES$  (fused-resource.fuse (lazy-core DH0-sample) lazy-rest) (basic-core-sinit,  
 basic-rest-sinit)  
 <proof>

**lemma** connect-real:  $\text{connect } D (\text{obsf-resource real-resource}) = \text{connect } D (\text{obsf-resource}$   
 $(RES (\text{fused-resource.fuse (lazy-core DH0-sample) lazy-rest) (\text{basic-core-sinit, ba-}$   
 $\text{sic-rest-sinit})))$   
 <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**  $\mathcal{I}$ -auth1-rest  
**and**  $\mathcal{I}$ -auth2-rest

**assumes**

$WT$ -auth1-rest [WT-intro]:  $WT$ -rest  $\mathcal{I}$ -adv-rest1  $\mathcal{I}$ -usr-rest1  $\mathcal{I}$ -auth1-rest auth1-rest  
**and**

$WT$ -auth2-rest [WT-intro]:  $WT$ -rest  $\mathcal{I}$ -adv-rest2  $\mathcal{I}$ -usr-rest2  $\mathcal{I}$ -auth2-rest auth2-rest  
**begin**

**theorem** secure:

**defines**  $\mathcal{I}$ -real  $\equiv ((\mathcal{I}$ -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  $\equiv (\mathcal{I}$ -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  $\equiv \mathcal{I}$ -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<sub>C</sub>) ⊙ lassocr<sub>C</sub>))  
 I-real I-ideal 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  $\mathcal{G} :: \text{security} \Rightarrow \text{'grp cyclic-group}$   
 assumes *diffie-hellman* [locale-witness]:  $\bigwedge \eta. \text{diffie-hellman } (\mathcal{G} \ \eta)$   
**begin**

**sublocale** *diffie-hellman*  $\mathcal{G} \ \eta$  **for**  $\eta$  ⟨proof⟩

**definition** *real-resource'* **where** *real-resource' rest1 rest2*  $\eta = \text{real-resource TYPE(-) TYPE(-) } \eta \ (\text{rest1 } \eta) \ (\text{rest2 } \eta)$

**definition** *ideal-resource'* **where** *ideal-resource' rest1 rest2*  $\eta = \text{key.resource } \eta \ (\text{ideal-rest } (\text{rest1 } \eta)) \ (\text{rest2 } \eta)$

**definition** *sim'* **where** *sim'*  $\eta = (\text{let sim = CNV (sim-callee } \eta) \ \text{None in } ((\text{sim } |= 1_C) \odot \text{lassocr}_C))$

**context**

fixes

*auth1-rest* :: nat  $\Rightarrow$  ('auth1-s-rest, auth.event, 'auth1-iadv-rest, 'auth1-iusr-rest, 'auth1-oadv-rest, 'auth1-ousr-rest) rest-wstate **and**

*auth2-rest* :: nat  $\Rightarrow$  ('auth2-s-rest, auth.event, 'auth2-iadv-rest, 'auth2-iusr-rest, 'auth2-oadv-rest, 'auth2-ousr-rest) rest-wstate **and**

*I-adv-rest1* **and** *I-adv-rest2* **and** *I-usr-rest1* **and** *I-usr-rest2* **and** *I-auth1-rest* **and** *I-auth2-rest*

assumes

*WT-auth1-rest*:  $\bigwedge \eta. \text{WT-rest } (\text{I-adv-rest1 } \eta) \ (\text{I-usr-rest1 } \eta) \ (\text{I-auth1-rest } \eta) \ (\text{auth1-rest } \eta)$  **and**

*WT-auth2-rest*:  $\bigwedge \eta. \text{WT-rest } (\text{I-adv-rest2 } \eta) \ (\text{I-usr-rest2 } \eta) \ (\text{I-auth2-rest } \eta) \ (\text{auth2-rest } \eta)$

**begin**

**theorem** *secure*:

**defines** *I-real*  $\equiv \lambda \eta. ((\text{I-full } \oplus_{\mathcal{I}} (\text{I-full } \oplus_{\mathcal{I}} \text{I-uniform } (\text{auth.Inp-Fedit ' (carrier } (\mathcal{G} \ \eta))) \ \text{UNIV})) \oplus_{\mathcal{I}} (\text{I-full } \oplus_{\mathcal{I}} (\text{I-full } \oplus_{\mathcal{I}} \text{I-uniform } (\text{auth.Inp-Fedit ' (carrier } (\mathcal{G} \ \eta))) \ \text{UNIV}))) \oplus_{\mathcal{I}} (\text{I-adv-rest1 } \eta \oplus_{\mathcal{I}} \text{I-adv-rest2 } \eta)$

**and** *I-common*  $\equiv \lambda \eta. (\text{I-uniform UNIV } (\text{key.Out-Alice ' carrier } (\mathcal{G} \ \eta)) \oplus_{\mathcal{I}} \text{I-uniform UNIV } (\text{key.Out-Bob ' carrier } (\mathcal{G} \ \eta))) \oplus_{\mathcal{I}} ((\text{I-full } \oplus_{\mathcal{I}} \text{I-full}) \oplus_{\mathcal{I}} (\text{I-usr-rest1 } \eta \oplus_{\mathcal{I}} \text{I-usr-rest2 } \eta))$

**and** *I-ideal*  $\equiv \lambda \eta. \text{I-full } \oplus_{\mathcal{I}} (\text{I-full } \oplus_{\mathcal{I}} (\text{I-adv-rest1 } \eta \oplus_{\mathcal{I}} \text{I-adv-rest2 } \eta))$

**assumes** *DDH*: *negligible* ( $\lambda\eta$ . *ddh.advantage* ( $\mathcal{G}$   $\eta$ ) (*DH-adversary* *TYPE*(-) *TYPE*(-)  
 $\eta$  (*auth1-rest*  $\eta$ ) (*auth2-rest*  $\eta$ ) ( $\mathcal{A}$   $\eta$ )))  
**shows** *constructive-security-obsf'* (*real-resource'* *auth1-rest* *auth2-rest*) (*ideal-resource'*  
*auth1-rest* *auth2-rest*) *sim'*  $\mathcal{I}$ -*real*  $\mathcal{I}$ -*ideal*  $\mathcal{I}$ -*common*  $\mathcal{A}$   
 $\langle$ *proof* $\rangle$

**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*  $\mathcal{I}$ -*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*

**shows** *WT-rest* ( $\mathcal{I}$ -*full*  $\oplus_{\mathcal{I}}$  ( $\mathcal{I}$ -*adv-rest1*  $\oplus_{\mathcal{I}}$   $\mathcal{I}$ -*adv-rest2*)) (( $\mathcal{I}$ -*full*  $\oplus_{\mathcal{I}}$   $\mathcal{I}$ -*full*)  $\oplus_{\mathcal{I}}$   
( $\mathcal{I}$ -*usr-rest1*  $\oplus_{\mathcal{I}}$   $\mathcal{I}$ -*usr-rest2*))

( $\lambda(-, s)$ . *pred-prod* *I-auth1-rest* *I-auth2-rest*  $s$ ) (*ideal-rest* *auth1-rest* *auth2-rest*)

$\langle$ *proof* $\rangle$

**end**

**locale** *dh-otp = dh*: *diffie-hellman*  $\mathcal{G}$  + *otp*: *one-time-pad*  $\mathcal{L}$

**for**  $\mathcal{G}$  :: '*grp* *cyclic-group*

**and**  $\mathcal{L}$  :: '*grp* *boolean-algebra* +

**assumes** *carrier-G-L*: *carrier*  $\mathcal{G}$  = *carrier*  $\mathcal{L}$

**begin**

**theorem** *secure*:

**assumes** *WT-rest*  $\mathcal{I}$ -*adv-resta*  $\mathcal{I}$ -*usr-resta* *I-auth-rest* *auth-rest*

**and** *WT-rest*  $\mathcal{I}$ -*adv-rest1*  $\mathcal{I}$ -*usr-rest1* *I-auth1-rest* *auth1-rest*

**and** *WT-rest*  $\mathcal{I}$ -*adv-rest2*  $\mathcal{I}$ -*usr-rest2* *I-auth2-rest* *auth2-rest*

**shows**

*constructive-security-obsf*

( $1_C$   $\models$  *wiring-c1r22-c1r22* (*CNV* *otp.enc-callee* ()) (*CNV* *otp.dec-callee* ()))  $\models$

$1_C$   $\triangleright$

*fused-wiring*  $\triangleright$  *diffie-hellman.real-resource*  $\mathcal{G}$  *auth1-rest* *auth2-rest*  $\parallel$  *dh.auth.resource*

$auth\text{-}rest$   
 $(otp.sec.resource (otp.ideal\text{-}rest (dh.ideal\text{-}rest auth1\text{-}rest auth2\text{-}rest) auth\text{-}rest))$   
 $((1_C \odot$   
 $(parallel\text{-}wiring \odot ((let sim = CNV dh.sim\text{-}callee None in (sim \models 1_C) \odot$   
 $lassocr_C \models 1_C) \odot parallel\text{-}wiring) \odot$   
 $1_C) \odot$   
 $(otp.sim \models 1_C))$   
 $((((\mathcal{I}\text{-}full \oplus_{\mathcal{I}} (\mathcal{I}\text{-}full \oplus_{\mathcal{I}} \mathcal{I}\text{-}uniform (otp.sec.Inp\text{-}Fedit ' carrier \mathcal{G}) UNIV)) \oplus_{\mathcal{I}}$   
 $(\mathcal{I}\text{-}full \oplus_{\mathcal{I}} (\mathcal{I}\text{-}full \oplus_{\mathcal{I}} \mathcal{I}\text{-}uniform (otp.sec.Inp\text{-}Fedit ' carrier \mathcal{G}) UNIV)))$   
 $\oplus_{\mathcal{I}}$   
 $(\mathcal{I}\text{-}full \oplus_{\mathcal{I}} (\mathcal{I}\text{-}full \oplus_{\mathcal{I}} \mathcal{I}\text{-}uniform (otp.sec.Inp\text{-}Fedit ' carrier \mathcal{L}) UNIV))) \oplus_{\mathcal{I}}$   
 $((\mathcal{I}\text{-}adv\text{-}rest1 \oplus_{\mathcal{I}} \mathcal{I}\text{-}adv\text{-}rest2) \oplus_{\mathcal{I}} \mathcal{I}\text{-}adv\text{-}resta))$   
 $((\mathcal{I}\text{-}full \oplus_{\mathcal{I}} (\mathcal{I}\text{-}full \oplus_{\mathcal{I}} \mathcal{I}\text{-}uniform (otp.sec.Inp\text{-}Fedit ' carrier \mathcal{L}) UNIV)) \oplus_{\mathcal{I}}$   
 $((\mathcal{I}\text{-}full \oplus_{\mathcal{I}} (\mathcal{I}\text{-}adv\text{-}rest1 \oplus_{\mathcal{I}} \mathcal{I}\text{-}adv\text{-}rest2)) \oplus_{\mathcal{I}} \mathcal{I}\text{-}adv\text{-}resta))$   
 $((\mathcal{I}\text{-}uniform (otp.sec.Inp\text{-}Send ' carrier \mathcal{L}) UNIV \oplus_{\mathcal{I}} \mathcal{I}\text{-}uniform UNIV$   
 $(otp.sec.Out\text{-}Recv ' carrier \mathcal{L})) \oplus_{\mathcal{I}}$   
 $((\mathcal{I}\text{-}full \oplus_{\mathcal{I}} \mathcal{I}\text{-}full) \oplus_{\mathcal{I}} (\mathcal{I}\text{-}usr\text{-}rest1 \oplus_{\mathcal{I}} \mathcal{I}\text{-}usr\text{-}rest2)) \oplus_{\mathcal{I}} \mathcal{I}\text{-}usr\text{-}resta))$   
 $A (0 + (ddh.advantage \mathcal{G}$   
 $(diffie\text{-}hellman.DH\text{-}adversary \mathcal{G} auth1\text{-}rest auth2\text{-}rest$   
 $(absorb$   
 $(absorb A$   
 $(obsf\text{-}converter (1_C \models wiring\text{-}c1r22\text{-}c1r22 (CNV otp.enc\text{-}callee$   
 $() (CNV otp.dec\text{-}callee ()) \models 1_C)))$   
 $(obsf\text{-}converter$   
 $(fused\text{-}wiring \odot (1_C \models_{\infty} converter\text{-}of\text{-}resource (1_C \models 1_C \triangleright$   
 $dh.auth.resource auth\text{-}rest)))))) +$   
 $0))$   
 $\langle proof \rangle$   
**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](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 indifferenciability to constructive cryptography (and back). In *Theory of Cryptography Conference (TCC 2016), Proceedings, Part I*, volume 9985 of *LNCS*, pages 3–24. Springer, 2016.