

A Core Calculus for Equational Proofs of Distributed Cryptographic Protocols: Technical Report

September 11, 2022

1 IPDL Syntax

IPDL is built from three layers: *protocols* are networks of mutually interacting *reactions*, each of which is a simple monadic, probabilistic program that computes an *expression*. In the context of a protocol, a reaction operates on a unique *channel* and may read from other channels, thereby utilizing expressions computed by other reactions. The syntax and judgements of IPDL are outlined in Figures 1, 2, respectively, and are parameterized by a user-defined *signature* Σ :

Definition 1 (Signature). *An IPDL signature Σ is a finite collection of:*

- *type symbols* \mathbf{t} ;
- *typed function symbols* $\mathbf{f} : \tau \rightarrow \sigma$; and
- *typed distribution symbols* $\mathbf{d} : \tau \multimap \sigma$.

We have a minimal set of data types, including the unit type $\mathbf{1}$, Booleans, products, as well as arbitrary type symbols \mathbf{t} , drawn from the signature Σ . Expressions are used for non-probabilistic computations, and are standard. All values in IPDL are bitstrings of a length given by data types, so we annotate the operations $\mathbf{fst}_{\tau \times \sigma}$ and $\mathbf{snd}_{\tau \times \sigma}$ with the type of the pair to determine the index to split the pair into two; for readability we omit this subscript whenever appropriate. Function symbols \mathbf{f} must be declared in the signature Σ , and for a constant $\mathbf{f} : \mathbf{1} \rightarrow \tau$, we write \mathbf{f} in place of $\mathbf{f}()$. Substitutions $\theta : \Gamma_1 \rightarrow \Gamma_2$ between type contexts are standard.

As mentioned above, reactions are monadic programs which may return expressions, sample from distributions, read from channels, branch on a value of type **Bool**, and sequentially compose. Analogously to function symbols, distribution symbols \mathbf{d} must be declared in the signature Σ , and for a constant $\mathbf{d} : \mathbf{1} \multimap \tau$, we write **samp** \mathbf{d} instead of **samp** ($\mathbf{d}()$). For readability, we often omit the type of the bound variable in a sequential composition, and write $x \leftarrow \mathbf{read} \ c; R$ simply as $x \leftarrow c; R$ wherever appropriate. Protocols in IPDL are given by a simple but expressive syntax: channel assignment $o := R$ assigns the reaction R to channel o ; parallel composition $P \parallel Q$ allows P and Q to freely interact concurrently; and channel generation **new** $o : \tau$ in P creates a new, internal channel for use in P . *Embeddings* $\phi : \Delta_1 \rightarrow \Delta_2$ between channel contexts are injective, type-preserving mappings specifying how to rename channels in Δ_2 to fit in the larger context Δ_1 .

Data Types	τ, σ	$::=$	$\mathbf{1} \mid \mathbf{Bool} \mid \tau \times \tau \mid \mathbf{t}$
Expressions	e	$::=$	$x \mid () \mid \mathbf{true} \mid \mathbf{false} \mid \mathbf{f} \ e \mid (e, e) \mid \mathbf{fst}_{\tau \times \sigma} \ e \mid \mathbf{snd}_{\tau \times \sigma} \ e$
Channels	i, o, c		
Reactions	R, S	$::=$	$\mathbf{ret} \ e \mid \mathbf{samp} \ (\mathbf{d} \ e) \mid \mathbf{read} \ c \mid \mathbf{if} \ e \ \mathbf{then} \ R \ \mathbf{else} \ R \mid x : \sigma \leftarrow R; R$
Protocols	P, Q	$::=$	$o := R \mid P \parallel P \mid \mathbf{new} \ o : \tau \ \mathbf{in} \ P$
Channel Sets	I, O	$::=$	$\{c_1, \dots, c_n\}$
Type Contexts	Γ	$::=$	$\cdot \mid \Gamma, x : \tau$
Channel Contexts	Δ	$::=$	$\cdot \mid \Delta, c : \tau$

Figure 1: Syntax of IPDL.

Expression Typing	$\Gamma \vdash e : \tau$
Reaction Typing	$\Delta; \Gamma \vdash R : I \rightarrow \tau$
Protocol Typing	$\Delta \vdash P : I \rightarrow O$
Substitutions	$\theta : \Gamma_1 \rightarrow \Gamma_2$
Embeddings	$\phi : \Delta_1 \rightarrow \Delta_2$
Expression Equality	$\Gamma \vdash e_1 = e_2 : \tau$
Reaction Equality	$\Delta; \Gamma \vdash R_1 = R_2 : I \rightarrow \tau$
Protocol Equality (Strict)	$\Delta \vdash P_1 = P_2 : I \rightarrow O$

Figure 2: Judgements of IPDL.

1.1 Typing

We restrict our attention to well-typed IPDL constructs. In addition to respecting data types, the typing judgments guarantee that all reads from channels in reactions are in scope, and that all channels are assigned at most one reaction in protocols. The typing $\Gamma \vdash e : \tau$ for expressions is standard, see Figure 3. Figure 4 shows the typing rules for reactions. Intuitively, $\Delta; \Gamma \vdash R : I \rightarrow \tau$ holds when R uses variables in Γ , reads from channels in I typed according to Δ , and returns a value of type τ . Figure 5 gives the typing rules for protocols: $\Delta \vdash P : I \rightarrow O$ holds when P uses inputs in I to assign reactions to the channels in O , all typed according to Δ .

Channel assignment $o := R$ has the type $I \rightarrow \{o\}$ when R is well-typed with an empty variable context, making use of inputs from I as well as of o . We allow R to read from its own output o to express divergence: the protocol $o := \text{read } o$ cannot reduce, which is useful for (conditionally) deactivating certain outputs. The typing rule for parallel composition $P \parallel Q$ states that P may use the outputs of Q as inputs while defining its own outputs, and vice versa. Importantly, the typing rules ensure that the outputs of P and Q are disjoint so that each channel carries a unique reaction. Finally, the rule for channel generation allows a protocol to select a fresh channel name o , assign it a type τ , and use it for internal computation and communication. Protocol typing plays a crucial role for modeling security. Simulation-based security in IPDL is modeled by the existence of a *simulator* **Sim** with an appropriate typing judgment, $\Delta \vdash \text{Sim} : I \rightarrow O$. Restricting the behavior of **Sim** to only use inputs along I is necessary to rule out trivial results (*e.g.*, **Sim** simply copies a secret from the specification).

$\boxed{\Gamma \vdash e : \tau}$					
$\frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau}$	$\frac{}{\Gamma \vdash () : 1}$	$\frac{}{\Gamma \vdash \text{true} : \text{Bool}}$	$\frac{}{\Gamma \vdash \text{false} : \text{Bool}}$	$\frac{f : \sigma \rightarrow \tau \in \Sigma \quad \Gamma \vdash e : \sigma}{\Gamma \vdash f \ e : \tau}$	
$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash (e_1, e_2) : \tau_1 \times \tau_2}$	$\frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \text{fst}_{\tau_1 \times \tau_2} \ e : \tau_1}$	$\frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \text{snd}_{\tau_1 \times \tau_2} \ e : \tau_2}$			

Figure 3: Typing for IPDL expressions.

1.2 Equational Logic

We now present the equational logic of IPDL. As mentioned above, the logic is divided into *exact* rules that establish semantic equivalences between protocols, and *approximate* rules that are used to discharge indistinguishability assumptions.

$$\boxed{\Delta; \Gamma \vdash R : I \rightarrow \tau}$$

$$\frac{\Gamma \vdash e : \tau}{\Delta; \Gamma \vdash \text{ret } e : I \rightarrow \tau} \quad
\frac{d : \sigma \rightarrow \tau \in \Sigma \quad \Gamma \vdash e : \sigma}{\Delta; \Gamma \vdash \text{samp } (d \ e) : I \rightarrow \tau} \quad
\frac{(i : \tau) \in \Delta \quad i \in I}{\Delta; \Gamma \vdash \text{read } i : I \rightarrow \tau}$$

$$\frac{\Gamma \vdash e : \text{Bool} \quad \Delta; \Gamma \vdash R_1 : I \rightarrow \tau \quad \Delta; \Gamma \vdash R_2 : I \rightarrow \tau}{\Delta; \Gamma \vdash \text{if } e \text{ then } R_1 \text{ else } R_2 : I \rightarrow \tau} \quad
\frac{\Delta; \Gamma \vdash R : I \rightarrow \sigma \quad \Delta; \Gamma, x : \sigma \vdash S : I \rightarrow \tau}{\Delta; \Gamma \vdash (x : \sigma \leftarrow R; S) : I \rightarrow \tau}$$

Figure 4: Typing for IPDL reactions.

$$\boxed{\Delta \vdash P : I \rightarrow O}$$

$$\frac{o : \tau \in \Delta \quad o \notin I \quad \Delta; \cdot \vdash R : I \cup \{o\} \rightarrow \tau}{\Delta \vdash (o := R) : I \rightarrow \{o\}} \quad
\frac{\Delta \vdash P : I \cup O_2 \rightarrow O_1 \quad \Delta \vdash Q : I \cup O_1 \rightarrow O_2}{\Delta \vdash P \parallel Q : I \rightarrow O_1 \cup O_2}$$

$$\frac{\Delta, o : \tau \vdash P : I \rightarrow O \cup \{o\}}{\Delta \vdash (\text{new } o : \tau \text{ in } P) : I \rightarrow O}$$

Figure 5: Typing for IPDL protocols.

1.2.1 Exact Equivalences

The bulk of the reasoning in IPDL is done using exact equivalences. At the expression level, we assume an ambient finite set of axioms of the form $\Gamma \vdash e_1 = e_2 : \tau$, where $\Gamma \vdash e_1 : \tau$ and $\Gamma \vdash e_2 : \tau$. The rules for expression equality are standard, see Figure 6.

At the reaction level, we analogously assume an ambient finite set of axioms of the form $\Delta; \Gamma \vdash R_1 = R_2 : I \rightarrow \tau$, where $\Delta; \Gamma \vdash R_1 : I \rightarrow \tau$ and $\Delta; \Gamma \vdash R_2 : I \rightarrow \tau$. The rules for reaction equality, shown in Figures 7, 8, ensure in particular that reactions form a *commutative monad*: we have

$$(x \leftarrow R_1; y \leftarrow R_2; S(x, y)) = (y \leftarrow R_2; x \leftarrow R_1; S(x, y))$$

whenever R_2 does not depend on x . All expected equivalences for commutative monads hold for reactions, including the usual monad laws and congruence of equivalence under monadic bind. The SAMP-PURE rule allows us to drop an unused sampling, and the READ-DET rule allows us to replace two reads from the same channel by a single one. The rules IF-LEFT, IF-RIGHT, and IF-EXT allow us to manipulate conditionals.

At the protocol level, we similarly assume an ambient finite set of axioms of the form $\Delta \vdash P_1 = P_2 : I \rightarrow O$, where $\Delta \vdash P_1 : I \rightarrow O$ and $\Delta \vdash P_2 : I \rightarrow O$. We use these axioms to specify user-defined functional assumptions, *e.g.*, the correctness of decryption. Exact protocol equivalences allow us to reason about communication between subprotocols and functional correctness, and to simplify intermediate computations. We will see later that exact equivalence implies the existence of a *bisimulation* on protocols, which in turn implies perfect computational indistinguishability against an arbitrary distinguisher. The rules for the exact equivalence of protocols are in Figures 9, 10; we now describe them informally.

The COMP-NEW rule allows us to permute parallel composition and the creation of a new channel, and the same as *scope extrusion* in process calculi [1]. The ABSORB-LEFT rule allows us to discard a component in a parallel composition if it has no outputs; this allows us to eliminate internal channels once they are no longer used. The DIVERGE rule allows us to simplify diverging reactions: if a channel reads from itself and continues as an arbitrary reaction R , then we can safely discard R as we will never reach it in the first place. The three (un)folding rules FOLD-IF-LEFT, FOLD-IF-RIGHT, and FOLD-BIND allow us to simplify composite reactions by bringing their components into the protocol level as separate internal channels. The rule SUBSUME states that channel dependency is transitive: if we depend on o_1 , and o_1 in turn depends on o_0 , then we also depend on o_0 , and this dependency

can be made explicit. The SUBST rule allows us to inline certain reactions into **read** commands. Inlining $o_1 := R_1$ into $o_2 := x \leftarrow \text{read } o_1; R_2$ is sound provided R_1 is *duplicable*: observing two independent results of evaluating R_1 is equivalent to observing the same result twice. This side condition is easily discharged whenever R_1 does not contain probabilistic sampling. Finally, the DROP rule allows dropping unused reads from channels in certain situations. Due to timing dependencies among channels, we only allow dropping reads from the channel $o_1 := R_1$ in the context of $o_2 := _ \leftarrow \text{read } o_1; R_2$ when we have that $(_ \leftarrow R_1; R_2) = R_2$. This side condition is met whenever all reads present in R_1 are also present in R_2 .

$$\boxed{\Gamma \vdash e_1 = e_2 : \tau}$$

$$\begin{array}{c}
\frac{\Gamma \vdash e : \tau}{\Gamma \vdash e = e : \tau} \text{REFL} \qquad \frac{\Gamma \vdash e_1 = e_2 : \tau}{\Gamma \vdash e_2 = e_1 : \tau} \text{SYM} \qquad \frac{\Gamma \vdash e_1 = e_2 : \tau \quad \Gamma \vdash e_2 = e_3 : \tau}{\Gamma \vdash e_1 = e_3 : \tau} \text{TRANS} \\
\\
\frac{\Gamma \vdash e_1 = e_2 : \tau \text{ axiom}}{\Gamma \vdash e_1 = e_2 : \tau} \text{AXIOM} \qquad \frac{\theta : \Gamma_1 \rightarrow \Gamma_2 \quad \Gamma_2 \vdash e_1 = e_2 : \tau}{\Gamma_1 \vdash \theta^*(e_1) = \theta^*(e_2) : \tau} \text{SUBST} \\
\\
\frac{\mathbf{f} : \sigma \rightarrow \tau \in \Sigma \quad \Gamma \vdash e = e' : \sigma}{\Gamma \vdash \mathbf{f} \ e = \mathbf{f} \ e' : \tau} \text{APP-CONG} \qquad \frac{\Gamma \vdash e_1 = e'_1 : \tau_1 \quad \Gamma \vdash e_2 = e'_2 : \tau_2}{\Gamma \vdash (e_1, e_2) = (e'_1, e'_2) : \tau_1 \times \tau_2} \text{PAIR-CONG} \\
\\
\frac{\Gamma \vdash e = e' : \tau_1 \times \tau_2}{\Gamma \vdash \mathbf{fst}_{\tau_1 \times \tau_2} \ e = \mathbf{fst}_{\tau_1 \times \tau_2} \ e' : \tau_1} \text{FST-CONG} \qquad \frac{\Gamma \vdash e = e' : \tau_1 \times \tau_2}{\Gamma \vdash \mathbf{snd}_{\tau_1 \times \tau_2} \ e = \mathbf{snd}_{\tau_1 \times \tau_2} \ e' : \tau_2} \text{SND-CONG} \\
\\
\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash \mathbf{fst}_{\tau_1 \times \tau_2} \ (e_1, e_2) = e_1 : \tau_1} \text{FST-PAIR} \qquad \frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash \mathbf{snd}_{\tau_1 \times \tau_2} \ (e_1, e_2) = e_2 : \tau_2} \text{SND-PAIR} \\
\\
\frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash e = (\mathbf{fst}_{\tau_1 \times \tau_2} \ e, \mathbf{snd}_{\tau_1 \times \tau_2} \ e) : \tau_1 \times \tau_2} \text{PAIR-EXT} \qquad \frac{\Gamma \vdash e : 1}{\Gamma \vdash e = () : 1} \text{ONE-EXT}
\end{array}$$

Figure 6: Equality for IPDL expressions.

2 Maude Formalization

2.1 Normal Forms

We work with protocols that start with a list of declarations of internal channels, using **new**, followed by a parallel compositions of channel assignments. The reactions in these assignments can be transformed into a list of binds of the form $x : \tau \leftarrow \text{read}(c)$, called bind-read reactions, followed by a reaction without binds. The list of binds can be regarded as commutative, as two reactions with the same list of binds in different order are equivalent due to the reaction equivalence rule EXCH. Similarly, different order of declarations of internal channels gives equivalent protocols, by using the protocol equivalence rule NEW-EXCH. When writing equivalence proofs, we do not want to make the use of these rules explicit. Therefore, we introduce normal forms of reactions and protocols. The normal form of a reaction $\mathbf{NF}(L, R, O)$ consists of a commutative list L of bind-read reactions, a bind-free reaction R and a chosen order O of the names of the variables occurring in the binds in L . The latter will be used to determine how to turn the normal form of a reaction into a regular reaction. During equivalence proofs, we may obtain either arbitrary binds in L or reactions R that are not bind-free. This will be represented as a pre-normal-form, written $\mathbf{preNF}(L, R, O)$, which is a normal form without restrictions on the occurring reactions. The general strategy will be to transform pre-normal-forms to normal forms by rule applications. The normal form of a protocol $\mathbf{newNF}(L, P, O)$ consists of a commutative list L of declarations of internal channels, a protocol P that does not start with internal channel declarations and again a designated order O for the names of internal channels occurring in the declarations

$$\boxed{\Delta; \Gamma \vdash R_1 = R_2 : I \rightarrow \tau}$$

$$\begin{array}{c}
\frac{\Delta; \Gamma \vdash R : I \rightarrow \tau}{\Delta; \Gamma \vdash R = R : I \rightarrow \tau} \text{REFL} \qquad \frac{\Delta; \Gamma \vdash R_1 = R_2 : I \rightarrow \tau}{\Delta; \Gamma \vdash R_2 = R_1 : I \rightarrow \tau} \text{SYM} \\
\\
\frac{\Delta; \Gamma \vdash R_1 = R_2 : I \rightarrow \tau \quad \Delta; \Gamma \vdash R_2 = R_3 : I \rightarrow \tau}{\Delta; \Gamma \vdash R_1 = R_3 : I \rightarrow \tau} \text{TRANS} \qquad \frac{\Delta; \Gamma \vdash R_1 = R_2 : I \rightarrow \tau \text{ axiom}}{\Delta; \Gamma \vdash R_1 = R_2 : I \rightarrow \tau} \text{AXIOM} \\
\\
\frac{\theta : \Gamma_1 \rightarrow \Gamma_2 \quad \Delta; \Gamma_2 \vdash R_1 = R_2 : I \rightarrow \tau}{\Delta; \Gamma_1 \vdash \theta^*(R_1) = \theta^*(R_2) : \theta^*(I) \rightarrow \tau} \text{SUBST} \qquad \frac{\phi : \Delta_1 \rightarrow \Delta_2 \quad \Delta_2; \Gamma \vdash R_1 = R_2 : I \rightarrow \tau}{\Delta_1; \Gamma \vdash \phi^*(R_1) = \phi^*(R_2) : \phi^*(I) \rightarrow \tau} \text{EMBED} \\
\\
\frac{i \notin I \quad \Delta; \Gamma \vdash R_1 = R_2 : I \rightarrow \tau}{\Delta; \Gamma \vdash R_1 = R_2 : I \cup \{i\} \rightarrow \tau} \text{INPUT-UNUSED} \qquad \frac{\Gamma \vdash e = e' : \tau}{\Delta; \Gamma \vdash \text{ret } e = \text{ret } e' : I \rightarrow \tau} \text{CONG-RET} \\
\\
\frac{d : \sigma \twoheadrightarrow \tau \in \Sigma \quad \Gamma \vdash e = e' : \sigma}{\Delta; \Gamma \vdash \text{samp } (d \ e) = \text{samp } (d \ e') : I \rightarrow \tau} \text{CONG-SAMP} \\
\\
\frac{\Gamma \vdash e = e' : \text{Bool} \quad \Delta; \Gamma \vdash R_1 = R'_1 : I \rightarrow \tau \quad \Delta; \Gamma \vdash R_2 = R'_2 : I \rightarrow \tau}{\Delta; \Gamma \vdash (\text{if } e \text{ then } R_1 \text{ else } R_2) = (\text{if } e' \text{ then } R'_1 \text{ else } R'_2) : I \rightarrow \tau} \text{CONG-IF} \\
\\
\frac{\Delta; \Gamma \vdash R = R' : I \rightarrow \sigma \quad \Delta; \Gamma, x : \sigma \vdash S = S' : I \rightarrow \tau}{\Delta; \Gamma \vdash (x : \sigma \leftarrow R; S) = (x : \sigma \leftarrow R'; S') : I \rightarrow \tau} \text{CONG-BIND}
\end{array}$$

Figure 7: Equality for lPDL reactions. Additional rules are given in Figure 8.

in L . Since in both cases the lists are commutative, we can write equivalence rules for normal forms where we assume that an internal channel declaration or a bind-read reaction are first in the lists.

References

- [1] R. Milner, J. Parrow, and D. Walker, “A calculus of mobile processes, i,” *Information and Computation*, vol. 100, no. 1, pp. 1–40, 1992.

$$\boxed{\Delta; \Gamma \vdash R_1 = R_2 : I \rightarrow \tau}$$

$$\frac{\Gamma \vdash e : \sigma \quad \Delta; \Gamma, x : \sigma \vdash R : I \rightarrow \tau}{\Delta; \Gamma \vdash (x \leftarrow \text{ret } e; R) = R[x := e] : I \rightarrow \tau} \text{RET-BIND}$$

$$\frac{\Delta; \Gamma \vdash R : I \rightarrow \tau}{\Delta; \Gamma \vdash (x \leftarrow R; \text{ret } x) = R : I \rightarrow \tau} \text{BIND-RET}$$

$$\frac{\Delta; \Gamma \vdash R_1 : I \rightarrow \sigma_1 \quad \Delta; \Gamma, x_1 : \sigma_1 \vdash R_2 : I \rightarrow \sigma_2 \quad \Delta; \Gamma, x_2 : \sigma_2 \vdash S : I \rightarrow \tau}{\Delta; \Gamma \vdash (x_2 : \sigma_2 \leftarrow (x_1 : \sigma_1 \leftarrow R_1; R_2); S) = (x_1 : \sigma_1 \leftarrow R_1; x_2 : \sigma_2 \leftarrow R_2; S) : I \rightarrow \tau} \text{BIND-BIND}$$

$$\frac{\Delta; \Gamma \vdash R_1 : I \rightarrow \sigma_1 \quad \Delta; \Gamma \vdash R_2 : I \rightarrow \sigma_2 \quad \Delta; \Gamma, x_1 : \sigma_1, x_2 : \sigma_2 \vdash S : I \rightarrow \tau}{\Delta; \Gamma \vdash (x_1 : \sigma_1 \leftarrow R_1; x_2 : \sigma_2 \leftarrow R_2; S) = (x_2 : \sigma_2 \leftarrow R_2; x_1 : \sigma_1 \leftarrow R_1; S) : I \rightarrow \tau} \text{EXCH}$$

$$\frac{\text{d} : \sigma_1 \twoheadrightarrow \sigma_2 \in \Sigma \quad \Gamma \vdash e : \sigma_1 \quad \Delta; \Gamma \vdash R : I \rightarrow \tau}{\Delta; \Gamma \vdash (x : \sigma_2 \leftarrow \text{samp } (\text{d } e); R) = R : I \rightarrow \tau} \text{SAMP-PURE}$$

$$\frac{i : \sigma \in \Delta \quad i \in I \quad \Delta; \Gamma, x : \sigma, y : \sigma \vdash R : I \rightarrow \tau}{\Delta; \Gamma \vdash (x : \sigma \leftarrow \text{read } i; y : \sigma \leftarrow \text{read } i; R) = (x : \sigma \leftarrow \text{read } i; R[y := x]) : I \rightarrow \tau} \text{READ-DET}$$

$$\frac{\Delta; \Gamma \vdash R_1 : I \rightarrow \tau \quad \Delta; \Gamma \vdash R_2 : I \rightarrow \tau}{\Delta; \Gamma \vdash (\text{if true then } R_1 \text{ else } R_2) = R_1 : I \rightarrow \tau} \text{IF-LEFT}$$

$$\frac{\Delta; \Gamma \vdash R_1 : I \rightarrow \tau \quad \Delta; \Gamma \vdash R_2 : I \rightarrow \tau}{\Delta; \Gamma \vdash (\text{if false then } R_1 \text{ else } R_2) = R_2 : I \rightarrow \tau} \text{IF-RIGHT}$$

$$\frac{\Delta; \Gamma, x : \text{Bool} \vdash R : I \rightarrow \tau \quad \Gamma \vdash e : \text{Bool}}{\Delta; \Gamma \vdash (\text{if } e \text{ then } R[x := \text{true}] \text{ else } R[x := \text{false}]) = R[x := e] : I \rightarrow \tau} \text{IF-EXT}$$

Figure 8: Equality for IPDL reactions.

$$\boxed{\Delta \vdash P = Q : I \rightarrow O}$$

$$\frac{\Delta \vdash P : I \rightarrow O}{\Delta \vdash P = P : I \rightarrow O} \text{REFL} \qquad \frac{\Delta \vdash P_1 = P_2 : I \rightarrow O}{\Delta \vdash P_2 = P_1 : I \rightarrow O} \text{SYM}$$

$$\frac{\Delta \vdash P_1 = P_2 : I \rightarrow O \quad \Delta \vdash P_2 = P_3 : I \rightarrow O}{\Delta \vdash P_1 = P_3 : I \rightarrow O} \text{TRANS} \qquad \frac{\Delta \vdash P = Q : I \rightarrow O \text{ axiom}}{\Delta \vdash P = Q : I \rightarrow O} \text{AXIOM}$$

$$\frac{\phi : \Delta_1 \rightarrow \Delta_2 \quad \Delta_2 \vdash P = Q : I \rightarrow O}{\Delta_1 \vdash \phi^*(P) = \phi^*(Q) : \phi^*(I) \rightarrow \phi^*(O)} \text{EMBED} \qquad \frac{i \notin I \quad i \notin O \quad \Delta \vdash P_1 = P_2 : I \rightarrow O}{\Delta \vdash P_1 = P_2 : I \cup \{i\} \rightarrow O} \text{INPUT-UNUSED}$$

$$\frac{o : \tau \in \Delta \quad \Delta; \cdot \vdash R = R' : I \cup \{o\} \rightarrow \tau}{\Delta \vdash (o := R) = (o := R') : I \rightarrow \{o\}} \text{CONG-REACT}$$

$$\frac{\Delta \vdash P = P' : I \cup O_2 \rightarrow O_1 \quad \Delta \vdash Q : I \cup O_1 \rightarrow O_2}{\Delta \vdash P \parallel Q = P' \parallel Q : I \rightarrow O_1 \cup O_2} \text{CONG-COMP-LEFT}$$

$$\frac{\Delta, o : \tau \vdash P = P' : I \rightarrow O \cup \{o\}}{\Delta \vdash (\text{new } o : \tau \text{ in } P = (\text{new } o : \tau \text{ in } P')) : I \rightarrow O} \text{CONG-NEW}$$

$$\frac{\Delta \vdash P_1 : I \cup O_2 \rightarrow O_1 \quad \Delta \vdash P_2 : I \cup O_1 \rightarrow O_2}{\Delta \vdash P_1 \parallel P_2 = P_2 \parallel P_1 : I \rightarrow O_1 \cup O_2} \text{COMP-COMM}$$

$$\frac{\Delta \vdash P_1 : I \cup O_2 \cup O_3 \rightarrow O_1 \quad \Delta \vdash P_2 : I \cup O_1 \cup O_3 \rightarrow O_2 \quad \Delta \vdash P_3 : I \cup O_1 \cup O_2 \rightarrow O_3}{\Delta \vdash (P_1 \parallel P_2) \parallel P_3 = P_1 \parallel (P_2 \parallel P_3) : I \rightarrow O_1 \cup O_2 \cup O_3} \text{COMP-ASSOC}$$

$$\frac{\Delta, o_1 : \tau_1, o_2 : \tau_2 \vdash P : I \rightarrow O \cup \{o_1, o_2\}}{\Delta \vdash (\text{new } o_1 : \tau_1 \text{ in new } o_2 : \tau_2 \text{ in } P) = (\text{new } o_2 : \tau_2 \text{ in new } o_1 : \tau_1 \text{ in } P) : I \rightarrow O} \text{NEW-EXCH}$$

$$\frac{\Delta \vdash P : I \cup O_2 \rightarrow O_1 \quad \Delta, o : \tau \vdash Q : I \cup O_1 \rightarrow O_2 \cup \{o\}}{\Delta \vdash P \parallel (\text{new } o : \tau \text{ in } Q) = \text{new } o : \tau \text{ in } (P \parallel Q) : I \rightarrow O_1 \cup O_2} \text{COMP-NEW}$$

$$\frac{\Delta \vdash P : I \rightarrow O \quad \Delta \vdash Q : I \cup O \rightarrow \emptyset}{\Delta \vdash P \parallel Q = P : I \rightarrow O} \text{ABSORB-LEFT}$$

Figure 9: Exact equality for IPDL protocols. Additional rules are given in Figure 10.

$$\boxed{\Delta \vdash P = Q : I \rightarrow O}$$

$$\frac{o : \tau \in \Delta \quad o \notin I \quad \Delta; \cdot \vdash R : I \cup \{o\} \rightarrow \tau}{\Delta \vdash (o := x : \tau \leftarrow \text{read } o; R) = (o := \text{read } o) : I \rightarrow \{o\}} \text{DIVERGE}$$

$$\frac{o \notin I \quad o : \tau \in \Delta \quad \Delta; \cdot \vdash R : I \cup \{o\} \rightarrow \text{Bool} \quad \Delta; \cdot \vdash S_1 : I \cup \{o\} \rightarrow \tau \quad \Delta; \cdot \vdash S_2 : I \cup \{o\} \rightarrow \tau}{\Delta \vdash (\text{new } l : \tau \text{ in } o := x : \text{Bool} \leftarrow R; \text{ if } x \text{ then } \text{read } l \text{ else } S_2 \parallel l := S_1) = (o := x : \text{Bool} \leftarrow R; \text{ if } x \text{ then } S_1 \text{ else } S_2) : I \rightarrow \{o\}} \text{FOLD-IF-LEFT}$$

$$\frac{o \notin I \quad o : \tau \in \Delta \quad \Delta; \cdot \vdash R : I \cup \{o\} \rightarrow \text{Bool} \quad \Delta; \cdot \vdash S_1 : I \cup \{o\} \rightarrow \tau \quad \Delta; \cdot \vdash S_2 : I \cup \{o\} \rightarrow \tau}{\Delta \vdash (\text{new } r : \tau \text{ in } o := x : \text{Bool} \leftarrow R; \text{ if } x \text{ then } S_1 \text{ else } \text{read } r \parallel r := S_2) = (o := x : \text{Bool} \leftarrow R; \text{ if } x \text{ then } S_1 \text{ else } S_2) : I \rightarrow \{o\}} \text{FOLD-IF-RIGHT}$$

$$\frac{o \notin I \quad o : \tau \in \Delta \quad \Delta; \cdot \vdash R : I \cup \{o\} \rightarrow \tau \quad \Delta; x : \sigma \vdash S : I \cup \{o\} \rightarrow \tau}{\Delta \vdash (\text{new } c : \sigma \text{ in } o := x : \sigma \leftarrow \text{read } c; S \parallel c := R) = (o := x : \sigma \leftarrow R; S) : I \rightarrow \{o\}} \text{FOLD-BIND}$$

$$\frac{o_0 : \tau_0, o_1 : \tau_1, o_2 : \tau_2 \in \Delta \quad o_1 \neq o_2 \quad o_1, o_2 \notin I \quad o_0 \in I \cup \{o_1, o_2\} \quad \Delta; x_0 : \tau_0 \vdash R_1 : I \cup \{o_1, o_2\} \rightarrow \tau_1 \quad \Delta; x_1 : \tau_1 \vdash R_2 : I \cup \{o_1, o_2\} \rightarrow \tau_2}{\Delta \vdash (o_1 := x_0 : \tau_0 \leftarrow \text{read } o_0; R_1 \parallel o_2 := x_0 : \tau_0 \leftarrow \text{read } o_0; x_1 : \tau_1 \leftarrow \text{read } o_1; R_2) = (o_1 := x_0 : \tau_0 \leftarrow \text{read } o_0; R_1 \parallel o_2 := x_1 : \tau_1 \leftarrow \text{read } o_1; R_2) : I \rightarrow \{o_1, o_2\}} \text{SUBSUME}$$

$$\frac{o_1 \neq o_2 \quad o_1, o_2 \notin I \quad o_1 : \tau_1, o_2 : \tau_2 \in \Delta \quad \Delta; \cdot \vdash R_1 : I \cup \{o_1, o_2\} \rightarrow \tau_1 \quad \Delta; x_1 : \tau_1 \vdash R_2 : I \cup \{o_1, o_2\} \rightarrow \tau_2 \quad \Delta; \cdot \vdash (x_1 \leftarrow R_1; x'_1 \leftarrow R_1; \text{ret } x_1, x'_1) = (x_1 \leftarrow R_1; \text{ret } x_1, x_1) : I \cup \{o_1, o_2\} \rightarrow \tau_1 \times \tau_1}{\Delta \vdash (o_1 := R_1 \parallel o_2 := x_1 : \tau_1 \leftarrow \text{read } o_1; R_2) = (o_1 := R_1 \parallel o_2 := x_1 : \tau_1 \leftarrow R_1; R_2) : I \rightarrow \{o_1, o_2\}} \text{SUBST}$$

$$\frac{o_1 \neq o_2 \quad o_1, o_2 \notin I \quad o_1 : \tau_1, o_2 : \tau_2 \in \Delta \quad \Delta; \cdot \vdash R_1 : I \cup \{o_1, o_2\} \rightarrow \tau_1 \quad \Delta; \cdot \vdash R_2 : I \cup \{o_1, o_2\} \rightarrow \tau_2 \quad \Delta; \cdot \vdash (x_1 : \tau_1 \leftarrow R_1; R_2) = R_2 : I \cup \{o_1, o_2\} \rightarrow \tau_2}{\Delta \vdash (o_1 := R_1 \parallel o_2 := x_1 \leftarrow \text{read } o_1; R_2) = (o_1 := R_1 \parallel o_2 := R_2) : I \rightarrow \{o_1, o_2\}} \text{DROP}$$

Figure 10: Additional rules for exact equality of IPDL protocols. Distinguishing changes of equalities are highlighted in red.