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

September 13, 2022

Acknowledgement

This project was funded through the NGI Assure Fund, a fund established by NLnet with financial support from the European Commission's Next Generation Internet programme, under the aegis of DG Communications Networks, Content and Technology under grant agreement No 957073.

1 Syntax of IPDL

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* t*;*
- *typed function symbols* $\mathsf{f} : \tau \to \sigma$*; and*
- *typed distribution symbols* $\mathsf{d} : \tau \twoheadrightarrow \sigma$*.*

We have a minimal set of data types, including the unit type $\mathsf{1}$, Booleans, products, as well as arbitrary type symbols 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 $\mathsf{fst}_{\tau\times\sigma}$ and $\mathsf{snd}_{\tau\times\sigma}$ 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 f must be declared in the signature Σ, and for a constant $\mathsf{f} : \mathsf{1} \to \tau$, we write f in place of $\mathsf{f}\ \checkmark$. Substitutions $\theta : \Gamma_1 \to \Gamma_2$ 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 d must be declared in the signature Σ, and for a constant $\mathsf{d} : \mathsf{1} \twoheadrightarrow \tau$, we write $\mathsf{samp}\ \mathsf{d}$ instead of $\mathsf{samp}\ (\mathsf{d}\ \checkmark)$. For readability, we often omit the type of the bound variable in a sequential composition, and write $x \leftarrow \mathsf{read}\ c;\ R$ 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 $P \mid\mid Q$ allows P and Q to freely interact concurrently; and channel generation $\mathsf{new}\ o : \tau\ \mathsf{in}\ P$ creates a new, internal channel for use in P. *Embeddings* $\phi : \Delta_1 \to \Delta_2$ between channel contexts are injective, type-preserving mappings specifying how to rename channels in Δ_2 to fit in the larger context Δ_1.

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

Data Types	τ, σ	$::=$	$\mathsf{t} \mid \mathsf{1} \mid \mathsf{Bool} \mid \tau \times \tau$
Expressions	e	$::=$	$x \mid \checkmark \mid \mathsf{true} \mid \mathsf{false} \mid \mathsf{f}\ e \mid (e_1, e_2) \mid \mathsf{fst}_{\tau \times \sigma}\ e \mid \mathsf{snd}_{\tau \times \sigma}\ e$
Channels	i, o, c		
Reactions	R, S	$::=$	$\mathsf{ret}\ e \mid \mathsf{samp}\ (\mathsf{d}\ e) \mid \mathsf{read}\ c \mid \mathsf{if}\ e\ \mathsf{then}\ R_1\ \mathsf{else}\ R_2 \mid x : \sigma \leftarrow R;\ S$
Protocols	P, Q	$::=$	$o := R \mid P \mid\mid Q \mid \mathsf{new}\ o : \tau\ \mathsf{in}\ P$
Channel Sets	I, O	$::=$	$\{c_1, \ldots, c_n\}$
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	$\Delta;\ \Gamma \vdash R : I \to \tau$
Protocol Typing	$\Delta \vdash P : I \to O$
Substitutions	$\theta : \Gamma_1 \to \Gamma_2$
Embeddings	$\phi : \Delta_1 \to \Delta_2$
Expression Equality	$\Gamma \vdash e_1 = e_2 : \tau$
Reaction Equality	$\Delta;\ \Gamma \vdash R_1 = R_2 : I \to \tau$
Protocol Equality (Strict)	$\Delta \vdash P_1 = P_2 : I \to O$

Figure 2: Judgements of ipdl.

rules for reactions. Intuitively, $\Delta;\ \Gamma \vdash R : I \to \tau$ 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: $\Delta \vdash P : I \to O$ 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 $I \to \{o\}$ 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 $o :=$ read o cannot reduce, which is useful for (conditionally) deactivating certain outputs. The typing rule for parallel composition $P \mid\mid Q$ 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, $\Delta \vdash \mathsf{Sim} : I \to O$. 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).

$$\boxed{\Gamma \vdash e : \tau}$$

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau} \qquad \frac{}{\Gamma \vdash \checkmark : 1} \qquad \frac{}{\Gamma \vdash \mathsf{true} : \mathsf{Bool}} \qquad \frac{}{\Gamma \vdash \mathsf{false} : \mathsf{Bool}} \qquad \frac{\mathsf{f} : \sigma \to \tau \in \Sigma \qquad \Gamma \vdash e : \sigma}{\Gamma \vdash \mathsf{f}\ e : \tau}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash (e_1, e_2) : \tau_1 \times \tau_2} \qquad \frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \mathsf{fst}_{\tau_1 \times \tau_2}\ e : \tau_1} \qquad \frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \mathsf{snd}_{\tau_1 \times \tau_2}\ e : \tau_2}$$

Figure 3: Typing for IPDL expressions.

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

$$\frac{\Gamma \vdash e : \tau}{\Delta;\ \Gamma \vdash \mathsf{ret}\ e : I \to \tau} \qquad \frac{\mathsf{d} : \sigma \twoheadrightarrow \tau \in \Sigma \qquad \Gamma \vdash e : \sigma}{\Delta;\ \Gamma \vdash \mathsf{samp}\ (\mathsf{d}\ e) : I \to \tau} \qquad \frac{(i : \tau) \in \Delta \qquad i \in I}{\Delta;\ \Gamma \vdash \mathsf{read}\ i : I \to \tau}$$

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

Figure 4: Typing for IPDL reactions.

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

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

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

Figure 5: Typing for IPDL protocols.

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.

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 $\Delta;\ \Gamma \vdash R_1 = R_2 : I \to \tau$, where $\Delta;\ \Gamma \vdash R_1 : I \to \tau$ and $\Delta;\ \Gamma \vdash R_2 : I \to \tau$. The rules for reaction equality, shown in Figures 7, 8, ensure in particular that reactions form a *commutative monad*: we have

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

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 $\Delta \vdash P_1 = P_2 : I \to O$, where $\Delta \vdash P_1 : I \to O$ and $\Delta \vdash P_2 : I \to O$. 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 $o_2 := x \leftarrow \mathsf{read}\ o_1;\ R_2$ 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 $o_2 := \_ \leftarrow \mathsf{read}\ o_1;\ R_2$ 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.

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

$$\frac{\Gamma \vdash e : \tau}{\Gamma \vdash e = e : \tau}\ \textsc{refl} \qquad \frac{\Gamma \vdash e_1 = e_2 : \tau}{\Gamma \vdash e_2 = e_1 : \tau}\ \textsc{sym} \qquad \frac{\Gamma \vdash e_1 = e_2 : \tau \qquad \Gamma \vdash e_2 = e_3 : \tau}{\Gamma \vdash e_1 = e_3 : \tau}\ \textsc{trans}$$

$$\frac{\Gamma \vdash e_1 = e_2 : \tau\ \mathsf{axiom}}{\Gamma \vdash e_1 = e_2 : \tau}\ \textsc{axiom} \qquad \frac{\theta : \Gamma_1 \to \Gamma_2 \qquad \Gamma_2 \vdash e_1 = e_2 : \tau}{\Gamma_1 \vdash \theta^\star(e_1) = \theta^\star(e_2) : \tau}\ \textsc{subst}$$

$$\frac{\mathsf{f} : \sigma \to \tau \in \Sigma \qquad \Gamma \vdash e = e' : \sigma}{\Gamma \vdash \mathsf{f}\ e = \mathsf{f}\ e' : \tau}\ \textsc{app-cong} \qquad \frac{\Gamma \vdash e_1 = e_1' : \tau_1 \qquad \Gamma \vdash e_2 = e_2' : \tau_2}{\Gamma \vdash (e_1, e_2) = (e_1', e_2') : \tau_1 \times \tau_2}\ \textsc{pair-cong}$$

$$\frac{\Gamma \vdash e = e' : \tau_1 \times \tau_2}{\Gamma \vdash \mathsf{fst}_{\tau_1 \times \tau_2}\ e = \mathsf{fst}_{\tau_1 \times \tau_2}\ e' : \tau_1}\ \textsc{fst-cong} \qquad \frac{\Gamma \vdash e = e' : \tau_1 \times \tau_2}{\Gamma \vdash \mathsf{snd}_{\tau_1 \times \tau_2}\ e = \mathsf{snd}_{\tau_1 \times \tau_2}\ e' : \tau_2}\ \textsc{snd-cong}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash \mathsf{fst}_{\tau_1 \times \tau_2}\ (e_1, e_2) = e_1 : \tau_1}\ \textsc{fst-pair} \qquad \frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash \mathsf{snd}_{\tau_1 \times \tau_2}\ (e_1, e_2) = e_2 : \tau_2}\ \textsc{snd-pair}$$

$$\frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash e = \big(\mathsf{fst}_{\tau_1 \times \tau_2} e, \mathsf{snd}_{\tau_1 \times \tau_2} e\big) : \tau_1 \times \tau_2}\ \textsc{pair-ext} \qquad \frac{\Gamma \vdash e : \mathsf{1}}{\Gamma \vdash e = \checkmark : \mathsf{1}}\ \textsc{one-ext}$$

Figure 6: Equality for IPDL expressions.

2 Case Studies in IPDL

In this section, we briefly describe the case studies we have completed in IPDL so far. The full proofs are available in a separate document [2]. We demonstrate through lines of code that the proof effort of IPDL scales well with increasing protocol complexity, see Figure 11.

2.1 Communication Protocols

2.1.1 Secure Communication from CPA Security

In our first case study (Section 1 of [2]), we prove secure a communication protocol that constructs a secure communication channel from an authenticated one. The authenticated channels allow the adversary to observe in-flight messages and schedule delivery of them; in contrast, the secure communication channels allow the adversary to schedule messages but grant no access to the message contents. We note that in this setup, the ability of the

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

$$\frac{\Delta;\ \Gamma \vdash R : I \to \tau}{\Delta;\ \Gamma \vdash R = R : I \to \tau}\ \textsc{refl} \qquad \frac{\Delta;\ \Gamma \vdash R_1 = R_2 : I \to \tau}{\Delta;\ \Gamma \vdash R_2 = R_1 : I \to \tau}\ \textsc{sym}$$

$$\frac{\Delta;\ \Gamma \vdash R_1 = R_2 : I \to \tau \qquad \Delta;\ \Gamma \vdash R_2 = R_3 : I \to \tau}{\Delta;\ \Gamma \vdash R_1 = R_3 : I \to \tau}\ \textsc{trans} \qquad \frac{\Delta;\ \Gamma \vdash R_1 = R_2 : I \to \tau\ \mathsf{axiom}}{\Delta;\ \Gamma \vdash R_1 = R_2 : I \to \tau}\ \textsc{axiom}$$

$$\frac{\theta : \Gamma_1 \to \Gamma_2 \qquad \Delta;\ \Gamma_2 \vdash R_1 = R_2 : I \to \tau}{\Delta;\ \Gamma_1 \vdash \theta^\star(R_1) = \theta^\star(R_2) : \theta^\star(I) \to \tau}\ \textsc{subst} \qquad \frac{\phi : \Delta_1 \to \Delta_2 \qquad \Delta_2;\ \Gamma \vdash R_1 = R_2 : I \to \tau}{\Delta_1;\ \Gamma \vdash \phi^\star(R_1) = \phi^\star(R_2) : \phi^\star(I) \to \tau}\ \textsc{embed}$$

$$\frac{i \notin I \qquad \Delta;\ \Gamma \vdash R_1 = R_2 : I \to \tau}{\Delta;\ \Gamma \vdash R_1 = R_2 : I \cup \{i\} \to \tau}\ \textsc{input-unused} \qquad \frac{\Gamma \vdash e = e' : \tau}{\Delta;\ \Gamma \vdash \mathsf{ret}\ e = \mathsf{ret}\ e' : I \to \tau}\ \textsc{cong-ret}$$

$$\frac{\mathsf{d} : \sigma \twoheadrightarrow \tau \in \Sigma \qquad \Gamma \vdash e = e' : \sigma}{\Delta;\ \Gamma \vdash \mathsf{samp}\ (\mathsf{d}\ e) = \mathsf{samp}\ (\mathsf{d}\ e') : I \to \tau}\ \textsc{cong-samp}$$

$$\frac{\Gamma \vdash e = e' : \mathsf{Bool} \qquad \Delta;\ \Gamma \vdash R_1 = R_1' : I \to \tau \qquad \Delta;\ \Gamma \vdash R_2 = R_2' : I \to \tau}{\Delta;\ \Gamma \vdash \big(\mathsf{if}\ e\ \mathsf{then}\ R_1\ \mathsf{else}\ R_2\big) = \big(\mathsf{if}\ e'\ \mathsf{then}\ R_1'\ \mathsf{else}\ R_2'\big) : I \to \tau}\ \textsc{cong-if}$$

$$\frac{\Delta;\ \Gamma \vdash R = R' : I \to \sigma \qquad \Delta;\ \Gamma, x : \sigma \vdash S = S' : I \to \tau}{\Delta;\ \Gamma \vdash (x : \sigma \leftarrow R;\ S) = (x : \sigma \leftarrow R';\ S') : I \to \tau}\ \textsc{cong-bind}$$

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

adversary to delay the messages sent over the secure channel is inherited from the corresponding control of the adversary over the authenticated channel.

2.2 Oblivious Transfer Protocols

We next consider a particular Oblivious Transfer (OT) construction. This example (Section 2 of [2]) is proven secure in the *semi-honest* (or *honest-but-curious*) setting, where all parties operate correctly, but corrupt parties leak all private data to the adversary. We prove that the leaked values reveal no private information about the honest parties. To encode semi-honest corruption, we augment the protocols with appropriate *leakage* functions that forward to the adversary all values visible to the corrupted parties. In turn, the simulator must take as input the leakages in the ideal protocol (usually minimal), and output suitable leakages in the real protocol.

In (1-out-of-2) OT, Bob wishes to obtain exactly one of Alice's two messages, without revealing his choice [3]. Alice doesn't learn which message Bob asked for, while Bob doesn't learn the other of the two messages. The ideal functionality simply receives the two messages m_0, m_1 from the sender, the choice bit c from the receiver, and outputs m_i. We analyze the most interesting case when Bob is semi-honest and Alice is honest. Hence, the real-world leakages are derived solely from the input i coming from the receiver, and the output m_i coming from the ideal functionality, with no access to any information about message m_{1-i}.

2.3 Multi-Party Coin Flip Protocol

Our first large case study (Section 3 of [2]) is for a protocol that allows an arbitrary number of mutually distrusting parties to collaboratively generate fair randomness, due to Blum [4]. To do so, each party locally generates randomness, and commits it to all other parties. We assume an idealized commitment functionality that also bakes in a notion of broadcast, to prevent equivocation. Each party decommits their randomness once all other commitments have been collected; the output of the protocol is the Boolean sum of all decommitments.

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

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

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

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

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

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

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

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

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

Figure 8: Equality for IPDL reactions.

Unlike previous examples, this example is secure in the *malicious* model. We model malicious parties by assigning them a *shell*, which simply forwards all information between the protocol and the adversary.

3 Maude Formalization

3.1 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 $x : \tau \leftarrow \mathit{read}(c)$, 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 $\mathsf{NF}(L, R, O)$ 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 occuring 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 $\mathsf{preNF}(L, R, O)$, which is a normal form without restrictions on the occuring reactions. The general strategy will be to transform pre-normal-forms to normal forms by rule applications. The normal form of a protocol $\mathsf{newNF}(L, P, O)$ 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 occuring in the declarations 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.

[2] K. Sojakova and M. Codescu, "Selected IPDL Case Studies," 2022. https://github.com/kristinas/IPDL-Maude/blob/main/doc/case_studies.pdf.

[3] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game," in *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pp. 218–229, ACM, 1987.

[4] M. Blum, "Coin flipping by telephone a protocol for solving impossible problems," *ACM SIGACT News*, vol. 15, no. 1, pp. 23–27, 1983.

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

$$\frac{\Delta \vdash P : I \to O}{\Delta \vdash P = P : I \to O}\ \textsc{refl} \qquad \frac{\Delta \vdash P_1 = P_2 : I \to O}{\Delta \vdash P_2 = P_1 : I \to O}\ \textsc{sym}$$

$$\frac{\Delta \vdash P_1 = P_2 : I \to O \qquad \Delta \vdash P_2 = P_3 : I \to O}{\Delta \vdash P_1 = P_3 : I \to O}\ \textsc{trans} \qquad \frac{\Delta \vdash P = Q : I \to O \text{ axiom}}{\Delta \vdash P = Q : I \to O}\ \textsc{axiom}$$

$$\frac{\phi : \Delta_1 \to \Delta_2 \qquad \Delta_2 \vdash P = Q : I \to O}{\Delta_1 \vdash \phi^\star(P) = \phi^\star(Q) : \phi^\star(I) \to \phi^\star(O)}\ \textsc{embed} \qquad \frac{i \notin I \qquad i \notin O \qquad \Delta \vdash P_1 = P_2 : I \to O}{\Delta \vdash P_1 = P_2 : I \cup \{i\} \to O}\ \textsc{input-unused}$$

$$\frac{o : \tau \in \Delta \qquad \Delta;\ \cdot \vdash R = R' : I \cup \{o\} \to \tau}{\Delta \vdash (o := R) = (o := R') : I \to \{o\}}\ \textsc{cong-react}$$

$$\frac{\Delta \vdash P = P' : I \cup O_2 \to O_1 \qquad \Delta \vdash Q : I \cup O_1 \to O_2}{\Delta \vdash P \mid\mid Q = P' \mid\mid Q : I \to O_1 \cup O_2}\ \textsc{cong-comp-left}$$

$$\frac{\Delta, o : \tau \vdash P = P' : I \to O \cup \{o\}}{\Delta \vdash (\text{new } o : \tau \text{ in } P = (\text{new } o : \tau \text{ in } P') : I \to O}\ \textsc{cong-new}$$

$$\frac{\Delta \vdash P_1 : I \cup O_2 \to O_1 \qquad \Delta \vdash P_2 : I \cup O_1 \to O_2}{\Delta \vdash P_1 \mid\mid P_2 = P_2 \mid\mid P_1 : I \to O_1 \cup O_2}\ \textsc{comp-comm}$$

$$\frac{\Delta \vdash P_1 : I \cup O_2 \cup O_3 \to O_1 \qquad \Delta \vdash P_2 : I \cup O_1 \cup O_3 \to O_2 \qquad \Delta \vdash P_3 : I \cup O_1 \cup O_2 \to O_3}{\Delta \vdash (P_1 \mid\mid P_2) \mid\mid P_3 = P_1 \mid\mid (P_2 \mid\mid P_3) : I \to O_1 \cup O_2 \cup O_3}\ \textsc{comp-assoc}$$

$$\frac{\Delta, o_1 : \tau_1, o_2 : \tau_2 \vdash P : I \to O \cup \{o_1, o_2\}}{\Delta \vdash \big(\text{new } o_1 : \tau_1 \text{ in new } o_2 : \tau_2 \text{ in } P\big) = \big(\text{new } o_2 : \tau_2 \text{ in new } o_1 : \tau_1 \text{ in } P\big) : I \to O}\ \textsc{new-exch}$$

$$\frac{\Delta \vdash P : I \cup O_2 \to O_1 \qquad \Delta, o : \tau \vdash Q : I \cup O_1 \to O_2 \cup \{o\}}{\Delta \vdash P \mid\mid (\text{new } o : \tau \text{ in } Q) = \text{new } o : \tau \text{ in } (P \mid\mid Q) : I \to O_1 \cup O_2}\ \textsc{comp-new}$$

$$\frac{\Delta \vdash P : I \to O \qquad \Delta \vdash Q : I \cup O \to \varnothing}{\Delta \vdash P \mid\mid Q = P : I \to O}\ \textsc{absorb-left}$$

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

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

$$\frac{o : \tau \in \Delta \qquad o \notin I \qquad \Delta;\ \cdot \vdash R : I \cup \{o\} \to \tau}{\Delta \vdash (o := x : \tau \leftarrow \mathsf{read}\ o;\ R) = (o := \mathsf{read}\ o) : I \to \{o\}}\ \textsc{Diverge}$$

$$\frac{\begin{array}{c} o \notin I \\ o : \tau \in \Delta \qquad \Delta;\ \cdot \vdash R : I \cup \{o\} \to \mathsf{Bool} \qquad \Delta;\ \cdot \vdash S_1 : I \cup \{o\} \to \tau \qquad \Delta;\ \cdot \vdash S_2 : I \cup \{o\} \to \tau \end{array}}{\begin{array}{c} \Delta \vdash \big(\mathsf{new}\ l : \tau\ \mathsf{in}\ o := x : \mathsf{Bool} \leftarrow R;\ \mathsf{if}\ x\ \mathsf{then}\ \mathsf{read}\ l\ \mathsf{else}\ S_2 \mid\mid l := S_1\big) = \\ \big(o := x : \mathsf{Bool} \leftarrow R;\ \mathsf{if}\ x\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\big) : I \to \{o\} \end{array}}\ \textsc{Fold-If-Left}$$

$$\frac{\begin{array}{c} o \notin I \\ o : \tau \in \Delta \qquad \Delta;\ \cdot \vdash R : I \cup \{o\} \to \mathsf{Bool} \qquad \Delta;\ \cdot \vdash S_1 : I \cup \{o\} \to \tau \qquad \Delta;\ \cdot \vdash S_2 : I \cup \{o\} \to \tau \end{array}}{\begin{array}{c} \Delta \vdash \big(\mathsf{new}\ r : \tau\ \mathsf{in}\ o := x : \mathsf{Bool} \leftarrow R;\ \mathsf{if}\ x\ \mathsf{then}\ S_1\ \mathsf{else}\ \mathsf{read}\ r \mid\mid r := S_2\big) = \\ \big(o := x : \mathsf{Bool} \leftarrow R;\ \mathsf{if}\ x\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\big) : I \to \{o\} \end{array}}\ \textsc{Fold-If-Right}$$

$$\frac{o \notin I \qquad o : \tau \in \Delta \qquad \Delta;\ \cdot \vdash R : I \cup \{o\} \to \tau \qquad \Delta;\ x : \sigma \vdash S : I \cup \{o\} \to \tau}{\Delta \vdash \big(\mathsf{new}\ c : \sigma\ \mathsf{in}\ o := x : \sigma \leftarrow \mathsf{read}\ c;\ S \mid\mid c := R\big) = \big(o := x : \sigma \leftarrow R;\ S\big) : I \to \{o\}}\ \textsc{Fold-Bind}$$

$$\frac{\begin{array}{c} o_1 \neq o_2 \qquad o_1, o_2 \notin I \qquad o_0 \in I \cup \{o_1, o_2\} \\ o_0 : \tau_0, o_1 : \tau_1, o_2 : \tau_2 \in \Delta \qquad \Delta;\ x_0 : \tau_0 \vdash R_1 : I \cup \{o_1, o_2\} \to \tau_1 \qquad \Delta;\ x_1 : \tau_1 \vdash R_2 : I \cup \{o_1, o_2\} \to \tau_2 \end{array}}{\begin{array}{c} \Delta \vdash \big(o_1 := x_0 : \tau_0 \leftarrow \mathsf{read}\ o_0;\ R_1 \mid\mid o_2 := x_0 : \tau_0 \leftarrow \mathsf{read}\ o_0;\ x_1 : \tau_1 \leftarrow \mathsf{read}\ o_1;\ R_2\big) = \\ \big(o_1 := x_0 : \tau_0 \leftarrow \mathsf{read}\ o_0;\ R_1 \mid\mid o_2 := x_1 : \tau_1 \leftarrow \mathsf{read}\ o_1;\ R_2\big) : I \to \{o_1, o_2\} \end{array}}\ \textsc{Subsume}$$

$$\frac{\begin{array}{c} o_1 \neq o_2 \qquad o_1, o_2 \notin I \qquad o_1 : \tau_1, o_2 : \tau_2 \in \Delta \\ \Delta;\ \cdot \vdash R_1 : I \cup \{o_1, o_2\} \to \tau_1 \qquad \Delta;\ x_1 : \tau_1 \vdash R_2 : I \cup \{o_1, o_2\} \to \tau_2 \\ \Delta;\ \cdot \vdash \big(x_1 \leftarrow R_1;\ x_1' \leftarrow R_1;\ \mathsf{ret}\ x_1, x_1'\big) = \big(x_1 \leftarrow R_1;\ \mathsf{ret}\ x_1, x_1\big) : I \cup \{o_1, o_2\} \to \tau_1 \times \tau_1 \end{array}}{\Delta \vdash \big(o_1 := R_1 \mid\mid o_2 := x_1 : \tau_1 \leftarrow \mathsf{read}\ o_1;\ R_2\big) = \big(o_1 := R_1 \mid\mid o_2 := x_1 : \tau_1 \leftarrow R_1;\ R_2\big) : I \to \{o_1, o_2\}}\ \textsc{Subst}$$

$$\frac{\begin{array}{c} o_1 \neq o_2 \qquad o_1, o_2 \notin I \qquad o_1 : \tau_1, o_2 : \tau_2 \in \Delta \qquad \Delta;\ \cdot \vdash R_1 : I \cup \{o_1, o_2\} \to \tau_1 \\ \Delta;\ \cdot \vdash R_2 : I \cup \{o_1, o_2\} \to \tau_2 \qquad \Delta;\ \cdot \vdash (x_1 : \tau_1 \leftarrow R_1;\ R_2) = R_2 : I \cup \{o_1, o_2\} \to \tau_2 \end{array}}{\Delta \vdash \big(o_1 := R_1 \mid\mid o_2 := x_1 \leftarrow \mathsf{read}\ o_1;\ R_2\big) = \big(o_1 := R_1 \mid\mid o_2 := R_2\big) : I \to \{o_1, o_2\}}\ \textsc{Drop}$$

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

Case study	Lines of Code
A2S: CPA	239 LoC
OT: Pre-Processing	480 LoC
Multi-Party Coin Flip	2019 LoC

Figure 11: Case Studies in ipdl.