

BLUEBELL: An Alliance of Relational Lifting and Independence for Probabilistic Reasoning

(Extended Version)

JIALU BAO, Cornell University, NY, US

EMANUELE D’OSUALDO, MPI-SWS, Germany and University of Konstanz, Germany

AZADEH FARZAN, University of Toronto, Canada

We present BLUEBELL, a program logic for reasoning about probabilistic programs where unary and relational styles of reasoning come together to create new reasoning tools. Unary-style reasoning is very expressive and is powered by foundational mechanisms to reason about probabilistic behavior like *independence* and *conditioning*. The relational style of reasoning, on the other hand, naturally shines when the properties of interest *compare* the behavior of similar programs (e.g. when proving differential privacy) managing to avoid having to characterize the output distributions of the individual programs. So far, the two styles of reasoning have largely remained separate in the many program logics designed for the deductive verification of probabilistic programs. In BLUEBELL, we unify these styles of reasoning through the introduction of a new modality called “joint conditioning” that can encode and illuminate the rich interaction between *conditional independence* and *relational liftings*; the two powerhouses from the two styles of reasoning.

CCS Concepts: • **Theory of computation** → **Separation logic; Logic and verification; Probabilistic computation; Program verification**; • **Software and its engineering** → *Software verification*.

Additional Key Words and Phrases: Deductive Verification, Relational Logic, Conditional Independence, Relational Lifting, Weakest Precondition

This article is an extended version of:

Jialu Bao, Emanuele D’Osualdo, and Azadeh Farzan. 2025. BLUEBELL: An Alliance of Relational Lifting and Independence for Probabilistic Reasoning. *Proc. ACM Program. Lang.* 9, POPL, Article 58 (January 2025), 31 pages. <https://doi.org/10.1145/3704894>

1 Introduction

Probabilistic programs are pervasive, appearing as machine learned subsystems, implementations of randomized algorithms, cryptographic protocols, and differentially private components, among many more. Ensuring reliability of such programs requires formal frameworks in which correctness requirements can be formalized and verified for such programs. Similar to the history of classical program verification, a lot of progress in this has come in the form of program logics for probabilistic programs. In the program logic literature, there are two main styles of reasoning for probabilistic programs: *unary* and *relational*, depending on the nature of the property of interest. For instance, for differential privacy or cryptographic protocols correctness, the property of interest is naturally expressible relationally. In contrast, for example, specifying the expected cost of a randomized algorithm is naturally done in the unary style.

Unary goals are triples $\{P\} t \{Q\}$ where t is a probabilistic program, P and Q are the pre- and post-conditions, i.e. *predicates over distributions of stores*. Such triples assert that running t on an input store drawn from a distribution satisfying P results in a distribution over output stores which satisfies Q . Unary reasoning for probabilistic programs has made great strides, producing logics for reasoning about expectations [Aguirre et al. 2021; Kaminski 2019; Kaminski et al. 2016; Kozen 1983; Moosbrugger et al. 2022; Morgan et al. 1996] and probabilistic independence [Barthe et al. 2019]. DIBI [Bao et al. 2021] and Lilac [Li et al. 2023a], which are the most recent, made a strong case for

Authors’ Contact Information: Jialu Bao, jlb965@cornell.edu, Cornell University, Ithaca, NY, US; Emanuele D’Osualdo, emanuele.dosualdo@uni-konstanz.de, MPI-SWS, Saarland Informatics Campus, Germany and University of Konstanz, Germany; Azadeh Farzan, azadeh@cs.toronto.edu, University of Toronto, Toronto, Canada.

adding power to reason about conditioning and independence in one logic. Intuitively, conditioning on some random variable x allows to focus on the distribution of other variables assuming x is some deterministic outcome v ; two variables are (conditionally) independent if knowledge of one does not give any knowledge of the other (under conditioning). Lilac argued for (conditional) independence as the fundamental source of modularity in the probabilistic setting.

Relational goals, in contrast, specify a desired relation between the output distributions of two programs t_1 and t_2 , for example, that t_1 and t_2 produce the same output distribution. In principle, proving such goals can be approached in a unary style: if the output distributions can be characterized individually for each program, then they can be compared after the fact. More often than not, however, precisely characterizing the output distribution of a program can be extremely challenging. Relational program logics like pRHL [Barthe et al. 2009] and its successors [Aguirre et al. 2017; Barthe et al. 2015, 2009; Gregersen et al. 2024; Hsu 2017], allow for a different and often more advantageous strategy. The idea is to consider the two programs side by side, and analyse their code as if executed in lockstep. If the effect of a step on one side is “matched” by the corresponding step on the other side, then the overall outputs would also “match”. This way, the proof only shows that whatever is computed on one side, will be matched by a computation on the other, without having to characterise what the actual output is.

This idea of “matching” probabilistic steps is formalised in these logics via the notion of *couplings* [Barthe et al. 2015, 2009]. The two programs can be conceptually considered to execute in two “parallel universes”, where they are oblivious to each other’s randomness. It is therefore sound to pretend their executions draw samples from a common source of randomness (called a *coupling*) in any way that eases the argument, as long as the marginal distribution of the correlated runs in each universe coincides with the original one. For example, if both programs flip a fair coin, one can force the outcomes of the coin flips to be the same (or the opposite of each other, depending on which serves the particular line of argument better). Relating the samples in a specific way helps with relating the distributions step by step, to support a relational goal. Couplings, when applicable, permit relational logics to elegantly sidestep the need to characterize the output distributions precisely. As such, relational logics hit an ergonomic sweet spot in reasoning style by restricting the form of the proofs that can be carried out.

Consider, for example, the code in Fig. 1. The $\text{BelowMax}(x, S)$ procedure takes N samples from a non-empty set $S \subseteq \mathbb{Z}$, according to an (arbitrary) distribution $\mu_S : \mathbb{D}(S)$; if any of the samples is larger than the given input x it declares x to be below the maximum of S . The $\text{AboveMin}(x, S)$ approximates in the same way whether x is above the minimum of S . These are Monte Carlo style algorithms with a *false bias*: if the answer is false, they always correctly produce it, and if the answer is true, then they correctly classify it with a probability that depends on N (i.e., the number of samples). It is a well-known fact that Monte Carlo style algorithms can be composed. For example, BETW_SEQ runs $\text{BelowMax}(x, S)$ and $\text{AboveMin}(x, S)$ to produce a *false-biased* Monte Carlo algorithm for approximately deciding whether x lies *within* the extrema of S . Now, imagine a programmer proposed BETW , as a way of getting more mileage out of the number of samples drawn; both procedures take $2N$ samples, but BETW performs more computation for each sample. Such optimisations are not really concerned about what the precise output distribution of each code is, but rather that a *true* answer is produced with higher probability by BETW ; in other words, its *stochastic dominance* over BETW_SEQ .

A unary program logic has only one way of reasoning about this type of stochastic-dominance: it has to analyze each code in isolation, characterize its output distribution, and finally assert/prove that one dominates the other. In contrast, there is a natural *relational strategy* for proving this goal: we can match the N samples of BelowMax with N of the samples of BETW , and the N samples of

```

def BelowMax(x, S):      def AboveMin(x, S):      def BETW_SEQ(x, S):      def BETW(x, S):
  repeat N:              repeat N:              BelowMax(x, S);      repeat 2N:
    q ≈ μS                p ≈ μS                AboveMin(x, S);      s ≈ μS
    r := r || q ≥ x      l := l || p ≤ x      d := r && l          l := l || s ≤ x
                                r := r || s ≥ x
                                d := r && l

```

Fig. 1. A stochastic dominance example: composing Monte Carlo algorithms two different ways. All variables are initially 0, $N \in \mathbb{N}$ is some fixed constant, S is a set of integers, and μ_S is some given distribution of elements of S . Both $\text{BETW_SEQ}(x, S)$ and $\text{BETW}(x, S)$ approximately decide whether x lies within the extrema of S .

AboveMin with the remaining samples of BETW in lockstep, and for each of these aligned steps, BETW has more chances of turning 1 and r to 1 (and they can only increase).

Unary logics can express information about distributions with arbitrary levels of precision; yet none can encode the simple natural proof idea outlined above. This suggests an opportunity: Bring native relational reasoning support to an expressive unary logic, like Lilac. Such a logic can be based on assertions over distributions, offering the precision and expressiveness of unary logics while natively supporting relational reasoning. As a result, it would be able to encode the argument outlined above at the appropriate level of abstraction. To explore this idea, let us outline the central principle that we would need to import from relational reasoning: *relational lifting*.

Relational logics use variants of judgments of the form $\{R_1\}[1:t_1, 2:t_2]\{R_2\}$, where t_1 and t_2 are the two programs we are comparing and R_1 and R_2 are the relational pre- and post-conditions. R_1 and R_2 differ from unary assertions in two ways: first they are used to relate two distributions instead of constraining a single one. Second, they are predicates over *pairs of stores*, and not of distributions directly. Let us call predicates of this type “deterministic relations”. If R was a deterministic predicate over a single store, requiring it to hold with probability 1 would naturally lift it to a predicate $[R]$ over distributions of stores. When R is a deterministic relation between pairs of stores, its *relational lifting* $[R]$ relates two distributions over stores $\mu_1, \mu_2 : \mathbb{D}(\mathbb{S})$, if (1) there is a distribution over *pairs of stores* $\mu : \mathbb{D}(\mathbb{S} \times \mathbb{S})$ such that its marginal distributions on the first and second store coincide with μ_1 and μ_2 respectively, (i.e. μ is a *coupling* of μ_1 and μ_2) and (2) μ samples pairs of stores satisfying the relation R with probability 1. Such relational liftings can encode a variety of useful relations between distributions. For instance, let $R = (x\langle 1 \rangle = x\langle 2 \rangle)$ relate stores s_1 and s_2 if they both assign the same value to x ; then the lifting $[R]$ holds for two distributions $\mu_1, \mu_2 : \mathbb{D}(\mathbb{S})$ if and only if they induce the same distributions in x . Similarly, the lifting $[x\langle 1 \rangle \leq x\langle 2 \rangle]$ encodes stochastic dominance of the distribution of x in μ_2 over the one in μ_1 .

Relational proofs built out of relational lifting then work by using deterministic relations as assertions, and showing that a suitably coupled lockstep execution of the two programs satisfies each assertion with probability 1. To bring relational reasoning to unary logics, we want to preserve the fact that assertions are over distributions, and yet support relational lifting as the key abstraction to do relational reasoning. This new logic can equally be viewed as a relational logic with assertions over pairs of distributions (rather than over pairs of stores). With such a view, seeing relational lifting as one among many constructs to build assertions seems like a very natural, yet completely unexplored, idea.

What is entirely non-obvious is whether relational lifting works well as an abstraction together with the other key “unary” constructs, such as independence and conditioning, that are the source of expressive power of unary logics. The properties of couplings already provide a source of examples

for one way in which unary and relational facts might interact. For example, from a well-known property of couplings, we know that establishing $\lfloor x\langle 1 \rangle = x\langle 2 \rangle \rfloor$ implies that $x\langle 1 \rangle$ and $x\langle 2 \rangle$ are identically distributed; this can be expressed as an entailment:

$$\lfloor x\langle 1 \rangle = x\langle 2 \rangle \rfloor \dashv\vdash \exists \mu. x\langle 1 \rangle \sim \mu \wedge x\langle 2 \rangle \sim \mu \quad (1)$$

The equivalence says that establishing a coupling that can (almost surely) equate the values of $x\langle 1 \rangle$ and $x\langle 2 \rangle$, amounts to establishing that the two variables are identically distributed. The equivalence can be seen as a way to interface “unary” facts and relational liftings.

Probability theory is full of lemmas of this sort and it is clearly undesirable to admit any lemma that is needed for one proof or another as an axiom in the program logic. Can we have a logic in which they are derivable without having to abandon its nice abstractions? Can the two styles be interoperable at the level of the logic? In this paper, we provide an affirmative answer to this question by proposing a new program logic called BLUEBELL.

We propose that relational lifting does in fact have non-trivial and useful interactions with independence and conditioning. Remarkably, BLUEBELL’s development is unlocked by a more fundamental observation: once an appropriate notion of conditioning is defined in BLUEBELL, relational lifting and its laws can be derived from this foundational conditioning construct.

The key idea is a new characterization of relational lifting as a form of conditioning: whilst relational lifting is usually seen as a way to induce a relation over distributions from a deterministic relation, BLUEBELL sees it as a way to go from a tuple of distributions to a relation between the values of some conditioned variables. More precisely:

- We introduce a new *joint conditioning* modality in BLUEBELL which can be seen, in hindsight, as a natural way to condition when dealing with tuples of distributions.
- We show that joint conditioning can represent uniformly both, conditioning *à la* Lilac, and relational lifting as derived notions in BLUEBELL.
- We prove a rich set of general rules for joint conditioning, from which we can derive both known and novel proof principles for conditioning and for relational liftings in BLUEBELL.

Interestingly, our joint conditioning modality can replicate the same reasoning style of Lilac’s modality, while having a different semantics (and validating an overlapping but different set of rules as a result). This deviation in the semantics is a stepping stone for obtaining an adequate generalization to the n -ary case (unifying unary and binary as special cases). We expand on these ideas in Section 2, using a running example. More importantly, our joint conditioning enables BLUEBELL to

- Accommodate unary and relational reasoning in a fundamentally interoperable way: For instance, we showcase the interaction between lifting and conditioning in the derivation of our running example in Section 2.
- Illuminate known reasoning principles: For instance, we discuss how BLUEBELL emulates pRHL-style reasoning in Section 5.1.
- Propose new tools to build program proofs: For instance, we discuss out-of-order coupling of samples through *SEQ-SWAP* in Section 2.4.
- Enable the exploration of the theory of high-level constructs like relational lifting (via the laws of independence and joint conditioning): For instance, novel broadly useful rules *RL-MERGE* and *RL-CONVEX*, discussed in Section 2 can be derived within BLUEBELL.

All proofs, omitted details, and additional examples can be found in the Appendix.

2 A Tour of BLUEBELL

In this section we will highlight the main key ideas behind BLUEBELL, using a running example.

2.1 The Alliance

We work with a first-order imperative probabilistic programming language consisting of programs $t \in \mathbb{T}$ that mutate a variable store $s \in \mathbb{S}$ (i.e. a finite map from variable names \mathbb{X} to values \mathbb{V}). We only consider discrete distributions (but with possibly infinite support). In Fig. 2 we show a simple example adapted from [Barthe et al. 2019]: the `encrypt` procedure uses a fair coin flip to generate an encryption key k , generates a plaintext message in boolean variable m (using a coin flip with some bias p) and produces the ciphertext c by XORing the key and the message. A desired property of the program is that the ciphertext should be indistinguishable from an unbiased coin flip; as a binary triple:

```
def encrypt():
  k := Ber(1/2)
  m := Ber(p)
  c := k xor m
```

Fig. 2. One time pad.

$$\{\text{True}\}[1:\text{encrypt}(), 2:c \approx \text{Ber}(1/2)]\{\llbracket c\langle 1 \rangle = c\langle 2 \rangle \rrbracket\} \quad (2)$$

where we use the $\langle i \rangle$ notation to indicate the index of the program store that an expression references. In Section 5.2, we discuss a unary-style proof of this goal in BLUEBELL. Here, we focus on a relational argument, as a running example. The natural (relational) argument goes as follows. When computing the final XOR, if $m = 0$ then $c = k$, and if $m = 1$ then $c = \neg k$. Since both $k\langle 1 \rangle$ and $c\langle 2 \rangle$ are distributed as *unbiased* coins, they can be coupled either so that they get the same value, or so that they get opposite values (the marginals are the same). One or the other coupling must be established *conditionally* on $m\langle 1 \rangle$, to formalize this argument. Doing so in pRHL faces the problem that the logic is too rigid to permit one to condition on $m\langle 1 \rangle$ before $k\langle 1 \rangle$ is sampled; rather it forces one to establish a coupling of $k\langle 1 \rangle$ and $c\langle 2 \rangle$ right when the two samplings happen. This rigidity is a well-known limitation of relational logics, which we overcome by “immersing” relational lifting in a logic with assertions on distributions. Recent work [Geggersen et al. 2024] proposed workarounds based on ghost code for pre-sampling (see Section 6). We present a different solution based on framing, to the generic problem of out-of-order coupling, in Section 2.4.

Unconstrained by the pRHL assumption that every assertion has to be represented as a relational lifting, we observe three crucial components in the proof idea:

- (1) *Probabilistic independence* between the sampling of $k\langle 1 \rangle$ and $m\langle 1 \rangle$, which makes conditioning on $m\langle 1 \rangle$ preserve the distribution of $k\langle 1 \rangle$;
- (2) *Conditioning* to perform case analysis on the possible values of $m\langle 1 \rangle$;
- (3) *Relational lifting* to represent the existence of couplings imposing the desired correlation between $k\langle 1 \rangle$ and $c\langle 2 \rangle$.

Unary logics like Probabilistic Separation Logics (PSL) [Barthe et al. 2019] and Lilac explored how probabilistic independence can be represented as *separating conjunction*, obtaining remarkably expressive and elegant reasoning principles. In BLUEBELL, we import the notion of independence from Lilac: BLUEBELL’s assertions are interpreted over tuples of probability spaces \mathcal{P} , and $Q_1 * Q_2$ holds on \mathcal{P} if $\mathcal{P}(i)$ can be seen as the *independent product* of $\mathcal{P}_1(i)$ and $\mathcal{P}_2(i)$, for each i , such that the tuples \mathcal{P}_1 and \mathcal{P}_2 satisfy Q_1 and Q_2 respectively. This means that $x\langle 1 \rangle \sim \mu * y\langle 1 \rangle \sim \mu$ states that $x\langle 1 \rangle$ and $y\langle 1 \rangle$ are independent and identically distributed, as opposed to $x\langle 1 \rangle \sim \mu \wedge y\langle 1 \rangle \sim \mu$ which merely declares the two variables as identically distributed (but possibly correlated). For a unary predicate over stores R we write $\llbracket R\langle i \rangle \rrbracket$ to mean that the predicate R holds with probability 1 in the distribution at index i .

With these tools it is easy to get through the first two assignments of `encrypt` and the one on component `2` and get to a state satisfying the assertion

$$P = k\langle 1 \rangle \sim \text{Ber}_{1/2} * m\langle 1 \rangle \sim \text{Ber}_p * c\langle 2 \rangle \sim \text{Ber}_{1/2} \quad (3)$$

The next ingredient we need is conditioning. We introduce a new modality C_μ for conditioning, in the spirit of Lilac. The modality takes the form $C_\mu v. K(v)$ where μ is a distribution, v is a logical variable bound by the modality (ranging over the support of μ), and $K(v)$ is a family of assertions indexed by v . Before exploring the general meaning of the modality (which we do in Section 2.2), let us illustrate how we would represent conditioning on $m\langle 1 \rangle$ in our running example. We know $m\langle 1 \rangle$ is distributed as Ber_p ; conditioning on $m\langle 1 \rangle$ in BLUEBELL would give us an assertion of the form $C_{\text{Ber}_p} v. K(v)$ where v ranges over $\{0, 1\}$, and the assertions $K(0) = \llbracket m\langle 1 \rangle = 0 \rrbracket * P_0$ and $K(1) = \llbracket m\langle 1 \rangle = 1 \rrbracket * P_1$ (for some P_0, P_1) represent the properties of the distribution conditioned on $m\langle 1 \rangle = 0$ and $m\langle 1 \rangle = 1$, respectively. Intuitively, the assertion states that the current distribution satisfies $K(v)$ when conditioned on $m\langle 1 \rangle = v$. Semantically, a distribution μ_0 satisfies the assertion $C_{\text{Ber}_p} v. K(v)$ if there exists distributions κ_0, κ_1 such that κ_0 satisfies $K(0)$, κ_1 satisfies $K(1)$, and μ_0 is the convex combination $\mu_0 = p \cdot \kappa_1 + (1 - p) \cdot \kappa_0$. When $K(v)$ constrains, as in our case, the value of a variable (here $m\langle 1 \rangle$) to be v , the only κ_0 and κ_1 satisfying the above constraints are the distribution μ_0 conditioned on the variable being 0 and 1 respectively.

Combining independence and conditioning with the third ingredient, relational lifting $\llbracket R \rrbracket$ (we discuss more about how to define it in Section 2.2), we can now express with an assertion the desired conditional coupling we foreshadowed in the beginning:

$$Q = C_{\text{Ber}_p} v. \left(\llbracket m\langle 1 \rangle = v \rrbracket * \begin{cases} \llbracket k\langle 1 \rangle = c\langle 2 \rangle \rrbracket & \text{if } v = 0 \\ \llbracket k\langle 1 \rangle = \neg c\langle 2 \rangle \rrbracket & \text{if } v = 1 \end{cases} \right) \quad (4)$$

The idea is that we first condition on $m\langle 1 \rangle$ so that we can see it as the deterministic value v , and then we couple $k\langle 1 \rangle$ and $c\langle 2 \rangle$ differently depending on v .

To carry out the proof idea formally, we are left with two subgoals. The first is to formally prove the entailment $P \vdash Q$. Then, it is possible to prove that after the final assignment to c at index `1`, the program is in a state that satisfies $Q * \llbracket c\langle 1 \rangle = k\langle 1 \rangle \text{ xor } m\langle 1 \rangle \rrbracket$. To finish the proof we would need to prove that $Q * \llbracket c\langle 1 \rangle = k\langle 1 \rangle \text{ xor } m\langle 1 \rangle \rrbracket \vdash \llbracket c\langle 1 \rangle = c\langle 2 \rangle \rrbracket$. These missing steps need laws governing the interaction among independence, conditioning and relational lifting in this n -ary setting.

A crucial observation of BLUEBELL is that, by choosing an appropriate definition for the joint conditioning modality C_μ , relational lifting can be encoded as a form of conditioning. Consequently, the laws governing relational lifting can be derived from the more primitive laws for joint conditioning. Moreover, the interactions between relational lifting and independence can be derived through the primitive laws for the interactions between joint conditioning and independence.

2.2 Joint Conditioning and Relational Lifting

Now we introduce the *joint* conditioning modality and its general n -ary version. Given $\mu : \mathbb{D}(A)$ and a function $\kappa : A \rightarrow \mathbb{D}(B)$ (called a *Markov kernel*), define the distribution $\text{bind}(\mu, \kappa) : \mathbb{D}(B)$ as $\text{bind}(\mu, \kappa) = \lambda b. \sum_{a \in A} \mu(a) \cdot \kappa(a)(b)$ and $\text{return}(v) = \delta_v$. The **bind** operation represents a convex combination with coefficients in μ , while δ_v is the Dirac distribution, which assigns probability 1 to the outcome v . These operations form a monad with the distribution functor $\mathbb{D}(\cdot)$, a special case of the Giry monad [Giry 1982]. Given a distribution $\mu : \mathbb{D}(A)$, and a predicate $K(a)$ over pairs of distributions parametrized by values $a \in A$, we define $C_\mu a. K(a)$ to hold on some (μ_1, μ_2) if

$$\exists \kappa_1, \kappa_2. \forall i \in \{1, 2\}. \mu_i = \text{bind}(\mu, \kappa_i) \wedge \forall a \in \text{supp}(\mu). K(a) \text{ holds on } (\kappa_1(a), \kappa_2(a))$$

Namely, we decompose the pair (μ_1, μ_2) component wise into convex compositions of μ and some kernels κ_1, κ_2 , one per component. Then for every a with non-zero probability in μ , we require the predicate $K(a)$ to hold for the pair of distributions $(\kappa_1(a), \kappa_2(a))$. The definition naturally extends to any number of indices.

One powerful application of the joint conditioning modality is to encode relational liftings $[R]$. Imagine we want to express $[k\langle 1 \rangle = c\langle 2 \rangle]$. It suffices to assert that there exists some distribution $\mu : \mathbb{D}(\mathbb{V} \times \mathbb{V})$ over pairs of values such that $C_\mu(v_1, v_2). ([k\langle 1 \rangle = v_1] \wedge [c\langle 2 \rangle = v_2] \wedge \ulcorner v_1 = v_2 \urcorner)$, where $\ulcorner \varphi \urcorner$ denote the embedding of a pure fact φ (i.e. a meta-level statement) into the logic. Let us digest the formula step-by-step. $C_\mu(v_1, v_2). ([k\langle 1 \rangle = v_1] \wedge [c\langle 2 \rangle = v_2])$ conditions the distribution at index **1** on $k\langle 1 \rangle = v_1$ and conditions the distribution at index **2** on $c\langle 2 \rangle = v_2$; such simultaneous conditioning is possible only if μ projected to its first index, $\mu \circ \pi_1^{-1}$, is the marginal distribution of $k\langle 1 \rangle$ and μ projected to its second index, $\mu \circ \pi_2^{-1}$, is the marginal distribution of $c\langle 2 \rangle$. Thus, μ is a joint distribution – a.k.a. coupling – of the marginal distributions of $k\langle 1 \rangle$ and $c\langle 2 \rangle$. The full assertion $C_\mu(v_1, v_2). ([k\langle 1 \rangle = v_1] \wedge [c\langle 2 \rangle = v_2] \wedge \ulcorner v_1 = v_2 \urcorner)$ ensures that the coupling μ has non-zero probabilities only on pairs (v_1, v_2) where $v_1 = v_2$, and this is exactly the requirement of the relational lifting $[k\langle 1 \rangle = c\langle 2 \rangle]$.

The idea generalizes to arbitrary relation lifting $[R]$, which are encoded using assertions of the form $\exists \mu. C_\mu(\vec{v}_1, \vec{v}_2). ([\vec{x}\langle 1 \rangle = \vec{v}_1] \wedge [\vec{x}\langle 2 \rangle = \vec{v}_2] \wedge \ulcorner R(\vec{v}_1, \vec{v}_2) \urcorner)$. The encoding hinges on the crucial decision in the design of the joint conditioning modality of using the same distribution μ to decompose the distributions at all indices. Then, the assertion inside the conditioning can force μ to be a joint distribution of (marginal) distributions of program states at different indices; and it can further force μ to have non-zero probability only on pairs of program states that satisfy the relation R .

The remarkable fact is that our formulation of relational lifting directly explains:

- (1) How the relational lifting can be *established*: that is, by providing some joint distribution μ for $k\langle 1 \rangle$ and $c\langle 2 \rangle$ ensuring R (the relation being lifted) holds for their joint outcomes; and
- (2) How the relational lifting can be *used* in entailments: that is, it guarantees that if one conditions on the store, R holds between the (now deterministic) variables.

To make these definitions and connections come to fruition we need to study which laws are supported by the joint conditioning modality and whether they are expressive enough to reason about distributions pairs without having to drop down to the level of semantics.

2.3 The Laws of Joint Conditioning

We survey the key laws for joint conditioning in this section, and explore a vital consequence of defining both conditional independence and relational lifting based on the joint conditioning modality: the laws of both can be derived from a set of expressive laws about joint conditioning alone. To keep the exposition concrete, we focus on a small subset of laws that are enough to prove the example of Section 2.1. Let us focus first on proving:

$$k\langle 1 \rangle \sim \text{Ber}_{1/2} * m\langle 1 \rangle \sim \text{Ber}_p * c\langle 2 \rangle \sim \text{Ber}_{1/2} \vdash C_{\text{Ber}_p} v. \left([m\langle 1 \rangle = v] * \begin{cases} [k\langle 1 \rangle = c\langle 2 \rangle] & \text{if } v = 0 \\ [k\langle 1 \rangle = \neg c\langle 2 \rangle] & \text{if } v = 1 \end{cases} \right) \quad (5)$$

We need the following primitive laws of joint conditioning:

<p>C-UNIT-R</p> $E\langle i \rangle \sim \mu \dashv\vdash C_\mu v. [E\langle i \rangle = v]$	<p>C-FRAME</p> $P * C_\mu v. K(v) \vdash C_\mu v. (P * K(v))$	<p>C-CONS</p> $\frac{\forall v. K_1(v) \vdash K_2(v)}{C_\mu v. K_1(v) \vdash C_\mu v. K_2(v)}$
--	---	--

Section 2.3 can convert back and forth from ownership of an expression E at i distributed as μ , and the conditioning on μ that makes E look deterministic. Section 2.3 allows to bring inside conditioning any resource that is independent from it. Section 2.3 simply allows to apply entailments inside joint conditioning. We can use these laws to perform conditioning on $m\langle 1 \rangle$:

$$\begin{aligned} k\langle 1 \rangle &\sim \text{Ber}_{\frac{1}{2}} * m\langle 1 \rangle \sim \text{Ber}_p * c\langle 2 \rangle \sim \text{Ber}_{\frac{1}{2}} \\ \vdash k\langle 1 \rangle &\sim \text{Ber}_{\frac{1}{2}} * (C_{\text{Ber}_p} v. \lceil m\langle 1 \rangle = v \rceil) * c\langle 2 \rangle \sim \text{Ber}_{\frac{1}{2}} & (\text{C-UNIT-R}) \\ \vdash C_{\text{Ber}_p} v. (\lceil m\langle 1 \rangle = v \rceil * k\langle 1 \rangle &\sim \text{Ber}_{\frac{1}{2}} * c\langle 2 \rangle \sim \text{Ber}_{\frac{1}{2}}) & (\text{C-FRAME}) \end{aligned}$$

Here we use **C-UNIT-R** to convert ownership of $m\langle 1 \rangle$ into its conditioned form. Then we can bring the other independent variables inside the conditioning with **C-FRAME**. This derivation follows in spirit the way in which Lilac introduces conditioning, thus inheriting its ergonomic elegance. Our rules however differ from Lilac's in both form and substance. First, Lilac's rule for introducing conditioning (called C-INDEP), requires converting ownership of a variable into conditioning, and bringing some independent resources inside conditioning, as a single monolithic step. In BLUEBELL we accomplish this pattern as a combination of our **C-UNIT-R** and **C-FRAME**, which are independently useful. Specifically, **C-UNIT-R** is bidirectional, which makes it useful to recover unconditional facts from conditional ones. Furthermore, we recognize that **C-UNIT-R** is nothing but a reflection of the right unit law of the monadic structure of distributions (which we elaborate on in Section 4). This connection prompted us to provide rules that reflect the remaining monadic laws (left unit **C-UNIT-L** and associativity **C-FUSE**). It is noteworthy that these rules do not follow from Lilac's proofs: our modality has a different semantics, and our rules seamlessly apply to assertions of any arity.

To establish the conditional relational liftings of the entailment in (5), BLUEBELL provides a way to introduce relational liftings from ownership of the distributions of some variables:

$$\begin{array}{c} \text{COUPLING} \\ \hline \mu \circ \pi_1^{-1} = \mu_1 \quad \mu \circ \pi_2^{-1} = \mu_2 \quad \mu(R) = 1 \\ \hline x_1\langle 1 \rangle \sim \mu_1 * x_2\langle 2 \rangle \sim \mu_2 \vdash \llbracket R(x_1\langle 1 \rangle, x_2\langle 2 \rangle) \rrbracket \end{array}$$

The side conditions of the rule ask the prover to provide a coupling $\mu : \mathbb{D}(\mathbb{V} \times \mathbb{V})$ of $\mu_1 : \mathbb{D}(\mathbb{V})$ and $\mu_2 : \mathbb{D}(\mathbb{V})$, which assigns probability 1 to a (binary) relation R . If $x_1\langle 1 \rangle$ and $x_2\langle 2 \rangle$ are distributed as μ_1 and μ_2 , respectively, then the relational lifting of R holds between them (as witnessed by the existence of μ). Note that for the rule to apply, the two variables need to live in distinct indices.

Interestingly, **COUPLING** can be derived from the encoding of relational lifting and the laws of joint conditioning.

Remarkably, although the rule mirrors the step of coupling two samplings in a pRHL proof, it does not apply to the code doing the sampling itself, but to the assertions representing the effects of those samplings. This allows us to delay the forming of coupling to until all necessary information is available (here, the outcome of $m\langle 1 \rangle$). We can use **COUPLING** to prove both entailments:

$$k\langle 1 \rangle \sim \text{Ber}_{\frac{1}{2}} * c\langle 2 \rangle \sim \text{Ber}_{\frac{1}{2}} \vdash \llbracket k\langle 1 \rangle = c\langle 2 \rangle \rrbracket \text{ and } k\langle 1 \rangle \sim \text{Ber}_{\frac{1}{2}} * c\langle 2 \rangle \sim \text{Ber}_{\frac{1}{2}} \vdash \llbracket k\langle 1 \rangle = \neg c\langle 2 \rangle \rrbracket \quad (6)$$

In the first case we use the coupling which flips a single coin and returns the same outcome for both components, in the second we flip a single coin but return opposite outcomes. Thus we can now prove:

$$C_{\text{Ber}_p} v. \left(\lceil m\langle 1 \rangle = v \rceil * \left(k\langle 1 \rangle \sim \text{Ber}_{\frac{1}{2}} * c\langle 2 \rangle \sim \text{Ber}_{\frac{1}{2}} \right) \right) \vdash C_{\text{Ber}_p} v. \left(\lceil m\langle 1 \rangle = v \rceil * \begin{cases} \llbracket k\langle 1 \rangle = c\langle 2 \rangle \rrbracket & \text{if } v = 0 \\ \llbracket k\langle 1 \rangle = \neg c\langle 2 \rangle \rrbracket & \text{if } v = 1 \end{cases} \right)$$

by using **C-CONS**, and using the two couplings of (6) in the $v = 0$ and $v = 1$ respectively. Finally, the assignment to c in `encrypt` generates the fact $\lceil c\langle 1 \rangle = k\langle 1 \rangle \text{ xor } m\langle 1 \rangle \rceil$. By routine propagation of this fact we can establish $C_{\text{Ber}_p} v. \lfloor c\langle 1 \rangle = c\langle 2 \rangle \rfloor$. To get an unconditional lifting, we need a principle explaining the interaction between lifting and conditioning. BLUEBELL can derive the general rule:

$$\begin{array}{c} \text{RL-CONVEX} \\ C_\mu _ . \lfloor R \rfloor \vdash \lfloor R \rfloor \end{array}$$

which states that relational liftings are *convex*, i.e. closed under convex combinations.

RL-CONVEX is an instance of many rules on the interaction between relational lifting and the other connectives (conditioning in this case) that can be derived in BLUEBELL by exploiting the encoding of liftings as joint conditioning.

Let us see how this is done for **RL-CONVEX** based on two other rules of joint conditioning:

$$\begin{array}{c} \text{C-SKOLEM} \\ \frac{\mu : \mathbb{D}(\Sigma_A)}{C_\mu v. \exists x : X. Q(v, x) \vdash \exists f : A \rightarrow X. C_\mu v. Q(v, f(v))} \end{array} \quad \begin{array}{c} \text{C-FUSE} \\ C_\mu v. C_{\kappa(v)} w. K(v, w) \dashv\vdash C_{\mu \prec \kappa}(v, w). K(v, w) \end{array}$$

Section 2.3 is a primitive rule which follows from Skolemization of the implicit universal quantification used on v by the modality. Section 2.3 can be seen as a way to merge two nested conditioning or split one conditioning into two. The rule uses $\mu \prec \kappa \triangleq \lambda(v, w). \mu(v) \cdot \kappa(v)(w)$, a variant of **bind** that does not forget the intermediate v . Section 2.3 is an immediate consequence of two primitive rules (**C-ASSOC** and **C-UNASSOC**) that reflect the associativity of the **bind** operation.

To prove **RL-CONVEX** we start by unfolding the definition of relational lifting (we write $K(v)$ for the part of the encoding inside the conditioning):

$$\begin{array}{l} C_\mu v. \lfloor R \rfloor \dashv\vdash C_\mu v. \exists \hat{\mu}_0. C_{\hat{\mu}_0} w. K(w) \\ \vdash \exists \kappa. C_\mu v. C_{\kappa(v)} w. K(w) \quad (\text{C-SKOLEM}) \\ \vdash \exists \kappa. C_{\mu \prec \kappa}(v, w). K(w) \quad (\text{C-FUSE}) \\ \vdash \exists \hat{\mu}_1. C_{\hat{\mu}_1}(v, w). K(w) \vdash \lfloor R \rfloor \quad (\text{By def.}) \end{array}$$

The application of **C-SKOLEM** commutes the existential quantification of the joint distribution $\hat{\mu}_0$ and the outer modality. By **C-FUSE** we are able to merge the two modalities and obtain again something of the same form as the encoding of relational liftings.

2.4 Outside the Box of Relational Lifting

One of the well-known limitations of pRHL is that it requires a very strict structural alignment between the order of samplings to be coupled in the two programs. A common pattern that pRHL rules cannot handle is showing that reversing the order of execution of two blocks of code does not affect the output distribution, e.g. running $x := \text{Ber}(\frac{1}{2}); y := \text{Ber}(\frac{2}{3})$ versus $y := \text{Ber}(\frac{2}{3}); x := \text{Ber}(\frac{1}{2})$. In BLUEBELL, we can establish this pattern using a *derived* general rule:

$$\begin{array}{c} \text{SEQ-SWAP} \\ \frac{\{P_1\} [1: t_1, 2: t'_1] \{ \lfloor R_1 \rfloor \} \quad \{P_2\} [1: t_2, 2: t'_2] \{ \lfloor R_2 \rfloor \}}{\{P_1 * P_2\} [1: (t_1; t_2), 2: (t'_2; t'_1)] \{ \lfloor R_1 \wedge R_2 \rfloor \}} \end{array}$$

The rule assumes that the lifting of R_1 (resp. R_2) can be established by analyzing t_1 and t'_1 (t_2 and t'_2) side by side from precondition P_1 (P_2). The standard sequential rule of pRHL would force an alignment between the wrong pairs (t_1 with t'_2 , and t_2 with t'_1). Crucial to the soundness of the rule is the assumption (expressed by the precondition $P_1 * P_2$ in the conclusion) that P_1 and P_2 are

probabilistically independent.¹ In contrast, because pRHL lacks the construct of independence, it simply cannot express such a rule.

BLUEBELL's treatment of relational lifting enables the study of the interaction between lifting and independence, unlocking a novel solution for forfeiting strict structural similarities between components required by relational logics.

Two ingredients of BLUEBELL cooperate to prove **SEQ-SWAP**: the adoption of a *weakest precondition* (WP) formulation of triples (and associated rules) and a novel property of relational lifting. Let us start with WP. In BLUEBELL, a triple $\{P\} t \{Q\}$ is actually encoded as the entailment $P \vdash \mathbf{wp} \, t \{Q\}$. Here, P and Q are both assertions on n -nary tuples of distributions; and throughout, we use the bold t to denote an n -nary tuple of program terms. The formula $\mathbf{wp} \, t \{Q\}$ is a natural generalization of WP assertion to n -nary programs: $\mathbf{wp} \, t \{Q\}$ holds on a n -nary tuple of distributions μ , if the tuple of output distributions obtained by running each program in t on the corresponding component of μ , satisfies Q . BLUEBELL provides a number of rules for manipulating WP; here is a selection needed for deriving **SEQ-SWAP**:

$$\begin{array}{c}
 \text{WP-CONS} \\
 \frac{Q \vdash Q'}{\mathbf{wp} \, t \{Q\} \vdash \mathbf{wp} \, t \{Q'\}} \\
 \\
 \text{WP-SEQ} \\
 \mathbf{wp} \, [i:t] \{ \mathbf{wp} \, [i:t'] \{Q\} \} \vdash \mathbf{wp} \, [i:(t; t')] \{Q\}
 \end{array}
 \qquad
 \begin{array}{c}
 \text{WP-FRAME} \\
 P * \mathbf{wp} \, t \{Q\} \vdash \mathbf{wp} \, t \{P * Q\} \\
 \\
 \text{WP-NEST} \\
 \mathbf{wp} \, t_1 \{ \mathbf{wp} \, t_2 \{Q\} \} \dashv \vdash \mathbf{wp} \, (t_1 \cdot t_2) \{Q\}
 \end{array}$$

Section 2.4 are the usual consequence and framing rules of Separation Logic, in WP form. By adopting Lilac's measure-theoretic notion of independence as the interpretation for separating conjunction, we obtain a clean frame rule. Among the WP rules for program constructs, Section 2.4 takes care of sequential composition. Notably, we only need to state it for unary WPs, in contrast to other logics where supporting relational proofs requires building the lockstep strategy into the rules. We use the more flexible approach from the Logic for Hypertriple Composition (LHC) [D'Ossualdo et al. 2022], where a handful of arity-changing rules allow seamless integration of unary and relational judgments. One such rule is the **WP-NEST** rule, which establishes the equivalence of a WP on $t_1 \cdot t_2$, where (\cdot) is union of indexed tuples with disjoint indexes, and two nested WPs involving t_1 , and t_2 individually. This for instance allows us to lift the unary **WP-SEQ** to a binary lockstep rule:

$$\begin{array}{c}
 \frac{P \vdash \mathbf{wp} \, [1:t_1] \{ \mathbf{wp} \, [2:t_2] \{Q\} \} \quad Q' \vdash \mathbf{wp} \, [1:t'_1] \{ \mathbf{wp} \, [2:t'_2] \{Q\} \}}{P \vdash \mathbf{wp} \, [1:t_1] \{ \mathbf{wp} \, [2:t_2] \{ \mathbf{wp} \, [1:t'_1] \{ \mathbf{wp} \, [2:t'_2] \{Q\} \} \} \}} \quad \text{WP-CONS} \\
 \frac{P \vdash \mathbf{wp} \, [1:t_1] \{ \mathbf{wp} \, [1:t'_1] \{ \mathbf{wp} \, [2:t_2] \{ \mathbf{wp} \, [2:t'_2] \{Q\} \} \} \}}{P \vdash \mathbf{wp} \, [1:(t_1; t'_1)] \{ \mathbf{wp} \, [2:(t_2; t'_2)] \{Q\} \}} \quad \text{WP-NEST} \\
 \frac{P \vdash \mathbf{wp} \, [1:(t_1; t'_1)] \{ \mathbf{wp} \, [2:(t_2; t'_2)] \{Q\} \}}{P \vdash \mathbf{wp} \, [1:(t_1; t'_1), 2:(t_2; t'_2)] \{Q\}} \quad \text{WP-SEQ, WP-CONS} \\
 \text{WP-NEST}
 \end{array}$$

The crucial idea behind **SEQ-SWAP** is that the two programs t_1 and t_2 we want to swap rely on *independent* resources, and thus their effects are independent from each other. In Separation Logic this kind of reasoning is driven by framing; which is done through framing in Separation Logic: while executing t_1 , frame the resources needed for t_2 , which remain intact in the state left by t_1 .

¹In the full model of BLUEBELL, to ensure safe mutation, assertions also include “write/read permissions” on variables (in the “variables as resource”-style [Bornat et al. 2005]). In **SEQ-SWAP** the separation between P_1 and P_2 ensures, in addition to probabilistic independence, that if t_1 has write permissions on a variable x , t_2 does not have read permissions on it and viceversa (and analogously for t'_1 and t'_2). The full model incurs in some permissions bookkeeping, which we omit in this section for readability; Example 4.15 shows how to fill in the omitted details.

Multiple applications of **WP-FRAME** and other basic rules get us to the post-condition $\llbracket R_1 \rrbracket * \llbracket R_2 \rrbracket$, but we want to combine them into one relational lifting. This is accommodated by:

RL-MERGE

$$\llbracket R_1 \rrbracket * \llbracket R_2 \rrbracket \vdash \llbracket R_1 \wedge R_2 \rrbracket$$

We do not show the derivation here for space constraints, but essentially it consists in unfolding the encoding of lifting, and using **C-FRAME** and **C-FUSE** to merge the two joint conditioning modalities.

Using these rules we can construct the following derivation:

$$\begin{array}{c}
 \frac{P_1 \vdash \mathbf{wp} [1: t_1, 2: t'_1] \{ \llbracket R_1 \rrbracket \} \quad P_2 \vdash \mathbf{wp} [1: t_2, 2: t'_2] \{ \llbracket R_2 \rrbracket \}}{P_1 * P_2 \vdash \mathbf{wp} [1: t_1, 2: t'_1] \{ \llbracket R_1 \rrbracket \} * \mathbf{wp} [1: t_2, 2: t'_2] \{ \llbracket R_2 \rrbracket \}} \text{WP-FRAME, WP-NEST} \\
 \frac{P_1 * P_2 \vdash \mathbf{wp} [1: t_1] \{ \mathbf{wp} [1: t_2, 2: t'_2] \{ \llbracket R_2 \rrbracket \} * \mathbf{wp} [2: t'_1] \{ \llbracket R_1 \rrbracket \} \}}{P_1 * P_2 \vdash \mathbf{wp} [1: t_1] \{ \mathbf{wp} [1: t_2, 2: t'_2] \{ \llbracket R_2 \rrbracket \} * \mathbf{wp} [2: t'_1] \{ \llbracket R_1 \rrbracket \} \}} \text{WP-FRAME} \\
 \frac{P_1 * P_2 \vdash \mathbf{wp} [1: t_1] \{ \mathbf{wp} [1: t_2, 2: t'_2] \{ \llbracket R_2 \rrbracket \} * \mathbf{wp} [2: t'_1] \{ \llbracket R_1 \rrbracket \} \}}{P_1 * P_2 \vdash \mathbf{wp} [1: t_1] \{ \mathbf{wp} [1: t_2, 2: t'_2] \{ \llbracket R_2 \rrbracket \} * \mathbf{wp} [2: t'_1] \{ \llbracket R_1 \rrbracket \} \}} \text{WP-FRAME} \\
 \frac{P_1 * P_2 \vdash \mathbf{wp} [1: t_1] \{ \mathbf{wp} [1: t_2, 2: t'_2] \{ \mathbf{wp} [2: t'_1] \{ \llbracket R_1 \rrbracket * \llbracket R_2 \rrbracket \} \} \}}{P_1 * P_2 \vdash \mathbf{wp} [1: t_1] \{ \mathbf{wp} [1: t_2, 2: t'_2] \{ \mathbf{wp} [2: t'_1] \{ \llbracket R_1 \rrbracket * \llbracket R_2 \rrbracket \} \} \}} \text{WP-SEQ, WP-NEST} \\
 \frac{P_1 * P_2 \vdash \mathbf{wp} [1: (t_1; t_2)] \{ \mathbf{wp} [2: (t'_2; t'_1)] \{ \llbracket R_1 \rrbracket * \llbracket R_2 \rrbracket \} \}}{P_1 * P_2 \vdash \mathbf{wp} [1: (t_1; t_2)] \{ \mathbf{wp} [2: (t'_2; t'_1)] \{ \llbracket R_1 \rrbracket * \llbracket R_2 \rrbracket \} \}} \text{WP-NEST} \\
 \frac{P_1 * P_2 \vdash \mathbf{wp} [1: (t_1; t_2), 2: (t'_2; t'_1)] \{ \llbracket R_1 \rrbracket * \llbracket R_2 \rrbracket \}}{P_1 * P_2 \vdash \mathbf{wp} [1: (t_1; t_2), 2: (t'_2; t'_1)] \{ \llbracket R_1 \wedge R_2 \rrbracket \}} \text{RL-MERGE}
 \end{array}$$

We explain the proof strategy from bottom to top. We first apply **RL-MERGE** to the postcondition (thanks to **WP-CONS**). This step reduces the goal to proving the two relational liftings can be established independently from each other. Then we apply **WP-NEST** and **WP-SEQ** to separate the two indices, break the sequential compositions and recombine the two inner WPs. We then proceed by three applications of the **WP-FRAME** rule: the first brings $\llbracket R_2 \rrbracket$ out of the innermost WP; the second brings the WP on $[1: t'_1]$ outside the middle WP; the last brings the WP on $[1: t_2, 2: t'_2]$ outside the topmost WP. An application of Section 2.4 merges the resulting nested WPs on t_1 and t'_1 . We thus effectively reduced the problem to showing that the two WPs can be established independently, which was our original goal.

The **RL-MERGE** rule not only provides an elegant way of overcoming the long-standing alignments issue with constructing relational lifting, but also shows how fundamental the role of probabilistic independence is for compositional reasoning: the same rule with standard conjunction is unsound! Intuitively, if we just had $\llbracket R_1 \rrbracket \wedge \llbracket R_2 \rrbracket$, we would know there exist two couplings μ_1 and μ_2 , justifying $\llbracket R_1 \rrbracket$ and $\llbracket R_2 \rrbracket$ respectively, but the desired consequence $\llbracket R_1 \wedge R_2 \rrbracket$ requires the construction of a single coupling that justifies both relations at the same time. We can see this is not always possible by looking back at (6): for two fair coins we can establish $\llbracket k\langle 1 \rangle = c\langle 2 \rangle \rrbracket \wedge \llbracket k\langle 1 \rangle = \neg c\langle 2 \rangle \rrbracket$, but $\llbracket k\langle 1 \rangle = c\langle 2 \rangle \wedge k\langle 1 \rangle = \neg c\langle 2 \rangle \rrbracket$ is equivalent to false.

3 Preliminaries: Programs and Probability Spaces

To formally define the model of BLUEBELL and validate its rules, we introduce a number of preliminary notions. Our starting point is the measure-theoretic approach of [Li et al. 2023a] in defining probabilistic separation. We recall the main definitions here. The main additional assumption we make throughout is that the set of outcomes of distributions is countable.

Definition 3.1 (Probability spaces). Given a set of possible *outcomes* Ω , a σ -algebra $\mathcal{F} \in \mathbb{A}(\Omega)$ is a set of subsets of Ω that is closed under countable unions and complements, and such that $\Omega \in \mathcal{F}$. The *full* σ -algebra over Ω is $\Sigma_\Omega = \mathcal{P}(\Omega)$, the powerset of Ω . For $F \subseteq \mathcal{P}(\Omega)$, we write $\sigma(F) \in \mathbb{A}(\Omega)$ for the smallest σ -algebra containing F . Given $\mathcal{F} \in \mathbb{A}(\Omega)$, a *probability distribution* $\mu \in \mathbb{D}(\mathcal{F})$, is a countably additive function $\mu: \mathcal{F} \rightarrow [0, 1]$ with $\mu(\Omega) = 1$. The support of a distribution $\mu \in \mathbb{D}(\Sigma_\Omega)$

$\mathbb{E} \ni e ::= v \mid x \mid \varphi(\vec{e}) \quad \vec{e} ::= e_1, \dots, e_n \quad \varphi ::= + \mid - \mid < \mid \dots \quad d ::= \text{Ber} \mid \text{Unif} \mid \dots$
 $\mathbb{T} \ni t ::= x := e \mid x \approx d(\vec{e}) \mid \text{skip} \mid t_1; t_2 \mid \text{if } e \text{ then } t_1 \text{ else } t_2 \mid \text{repeat } e \text{ } t$

Fig. 3. Syntax of program terms.

is the set of outcomes with non-zero probability $\text{supp}(\mu) \triangleq \{a \in \Omega \mid \mu(a) > 0\}$, where $\mu(a)$ abbreviates $\mu(\{a\})$.

A *probability space* $\mathcal{P} \in \mathbb{P}(\Omega)$ is a pair $\mathcal{P} = (\mathcal{F}, \mu)$ of a σ -algebra $\mathcal{F} \in \mathbb{A}(\Omega)$ and a probability distribution $\mu \in \mathbb{D}(\mathcal{F})$. The *trivial* probability space $1_\Omega \in \mathbb{P}(\Omega)$ is the trivial σ -algebra $\{\Omega, \emptyset\}$ equipped with the trivial probability distribution. Given $\mathcal{F}_1 \subseteq \mathcal{F}_2$ and $\mu \in \mathbb{D}(\mathcal{F}_2)$, the distribution $\mu|_{\mathcal{F}_1} \in \mathbb{D}(\mathcal{F}_1)$ is the restriction of μ to \mathcal{F}_1 . The *extension pre-order* (\sqsubseteq) over probability spaces is defined as $(\mathcal{F}_1, \mu_1) \sqsubseteq (\mathcal{F}_2, \mu_2) \triangleq \mathcal{F}_1 \subseteq \mathcal{F}_2 \wedge \mu_1 = \mu_2|_{\mathcal{F}_1}$.

A function $f: \Omega_1 \rightarrow \Omega_2$ is *measurable on* $\mathcal{F}_1 \in \mathbb{A}(\Omega_1)$ and $\mathcal{F}_2 \in \mathbb{A}(\Omega_2)$ if $\forall X \in \mathcal{F}_2. f^{-1}(X) \in \mathcal{F}_1$. When $\mathcal{F}_2 = \Sigma_{\Omega_2}$ we simply say f is *measurable in* \mathcal{F}_1 .

Definition 3.2 (Product and union spaces). Given $\mathcal{F}_1 \in \mathbb{A}(\Omega_1)$, $\mathcal{F}_2 \in \mathbb{A}(\Omega_2)$, their product is the σ -algebra $\mathcal{F}_1 \otimes \mathcal{F}_2 \in \mathbb{A}(\Omega_1 \times \Omega_2)$ defined as $\mathcal{F}_1 \otimes \mathcal{F}_2 \triangleq \sigma(\{X_1 \times X_2 \mid X_1 \in \mathcal{F}_1, X_2 \in \mathcal{F}_2\})$, and their union is the σ -algebra $\mathcal{F}_1 \oplus \mathcal{F}_2 \triangleq \sigma(\mathcal{F}_1 \cup \mathcal{F}_2)$. The product of two probability distributions $\mu_1 \in \mathbb{D}(\mathcal{F}_1)$ and $\mu_2 \in \mathbb{D}(\mathcal{F}_2)$ is the distribution $(\mu_1 \otimes \mu_2) \in \mathbb{D}(\mathcal{F}_1 \otimes \mathcal{F}_2)$ defined by $(\mu_1 \otimes \mu_2)(X_1 \times X_2) = \mu_1(X_1)\mu_2(X_2)$ for all $X_1 \in \mathcal{F}_1, X_2 \in \mathcal{F}_2$.

Definition 3.3 (Independent product [Li et al. 2023a]). Given $(\mathcal{F}_1, \mu_1), (\mathcal{F}_2, \mu_2) \in \mathbb{P}(\Omega)$, their *independent product* is the probability space $(\mathcal{F}_1 \oplus \mathcal{F}_2, \mu) \in \mathbb{P}(\Omega)$ where for all $X_1 \in \mathcal{F}_1, X_2 \in \mathcal{F}_2$, $\mu(X_1 \cap X_2) = \mu_1(X_1) \cdot \mu_2(X_2)$. It is unique, if it exists [Li et al. 2023a, Lemma 2.3]. Let $\mathcal{P}_1 \otimes \mathcal{P}_2$ be the unique independent product of \mathcal{P}_1 and \mathcal{P}_2 when it exists, and be undefined otherwise.

Indexed tuples. To handle unary and higher-arity relational assertions in a uniform way, we consider finite sets of indices $I \subseteq \mathbb{N}$, and I -indexed tuples of values of type X , represented as (finite) functions X^I . We use boldface to range over such functions. The syntax $\mathbf{x} = [i_0: x_0, \dots, i_n: x_n]$ denotes the function $\mathbf{x} \in X^{\{i_0, \dots, i_n\}}$ with $\mathbf{x}(i_k) = x_k$. We often use comprehension-style notation e.g. $\mathbf{x} = [i: x_i \mid i \in I]$. For $\mathbf{x} \in A^I$ we let $|\mathbf{x}| \triangleq I$. Given some $\mathbf{x} \in A^I$ and some $J \subseteq I$, the operation $\mathbf{x} \setminus J \triangleq [i: \mathbf{x}(i) \mid i \in I \setminus J]$ removes the components with indices in J from \mathbf{x} .

Programs. We consider a simple first-order imperative language. We fix a finite set of *program variables* $x \in \mathbb{X}$ and countable set of *values* $v \in \mathbb{V} \triangleq \mathbb{Z}$ and define the *program stores* to be $s \in \mathbb{S} \triangleq \mathbb{X} \rightarrow \mathbb{V}$ (note that \mathbb{S} is countable).

Program *terms* $t \in \mathbb{T}$ are formed according to the grammar in Fig. 3. For simplicity, booleans are encoded by using $0 \in \mathbb{V}$ as false and any other value as true. We will use the events $\text{false} \triangleq \{0\}$ and $\text{true} \triangleq \{n \in \mathbb{V} \mid n \neq 0\}$. Programs use standard deterministic primitives φ , which are interpreted as functions $\llbracket \varphi \rrbracket: \mathbb{V}^n \rightarrow \mathbb{V}$, where n is the arity of φ . Expressions e are effect-free deterministic numeric expressions, and denote, as is standard, a function $\llbracket e \rrbracket: \mathbb{S} \rightarrow \mathbb{V}$, i.e. a random variable of $\Sigma_{\mathbb{S}}$. We write $\text{pvar}(e)$ for the set of program variables that occur in e . Programs can refer to some collection of known *discrete* distributions d , each allowing a certain number of parameters. Sampling assignments $x \approx d(\vec{v})$ sample from the distribution $\llbracket d \rrbracket(\vec{v}): \mathbb{D}(\Sigma_{\mathbb{V}})$. The distribution $\llbracket \text{Ber} \rrbracket(p) = \text{Ber}_p: \mathbb{D}(\Sigma_{\{0,1\}})$ is the Bernoulli distribution assigning probability p to outcome 1.

Similar to Lilac, we consider a simple iteration construct **repeat** e t which evaluates e to a value $n \in \mathbb{V}$ and, if $n > 0$, executes t in sequence n times. This means we only consider a subset of terminating programs. The semantics of programs is entirely standard and is defined in Section B. It associates each term t to a function $\llbracket t \rrbracket: \mathbb{D}(\Sigma_{\mathbb{S}}) \rightarrow \mathbb{D}(\Sigma_{\mathbb{S}})$ from distributions of input stores to distributions of output stores.

In the relational reasoning setting, one would consider multiple programs at the same time and relate their semantics. Following LHC [D'Ousualdo et al. 2022], we define *hyper-terms* as $t \in \mathbb{T}^J$ for some finite set of indices J . Let I be such that $|t| \subseteq I$; the semantics $\llbracket t \rrbracket_I: \mathbb{D}(\Sigma_S)^I \rightarrow \mathbb{D}(\Sigma_S)^I$ takes a I -indexed family of distributions as input and outputs another I -indexed family of distributions:

$$\llbracket t \rrbracket_I(\mu) \triangleq \lambda i. \text{ if } i \in |t| \text{ then } \llbracket t(i) \rrbracket(\mu(i)) \text{ else } \mu(i)$$

Note that the store distributions at indices in $I \setminus |t|$ are preserved as is. We omit I when it can be inferred from context. To refer to program variables in a specific component we use elements of $I \times \mathbb{X}$, writing $x\langle i \rangle$ for (i, x) .

4 The BLUEBELL Logic

We are now ready to define BLUEBELL's semantic model, and formally prove its laws.

4.1 A Model of (Probabilistic) Resources

As a model for our assertions we use a modern presentation of partial commutative monoids, adapted from [Krebbers et al. 2018], called “ordered unital resource algebras” (henceforth RA). Instead of a partial binary operation, RAs are equipped with a *total* binary operation and a predicate \mathcal{V} indicating which elements of the carrier are considered *valid* resources. Partiality of the operation manifests as mapping some combinations of arguments to invalid elements.

Definition 4.1 (Ordered Unital Resource Algebra). An *ordered unital resource algebra* (RA) is a tuple $(M, \preceq, \mathcal{V}, \cdot, \varepsilon)$ where $\preceq: M \times M \rightarrow \text{Prop}$ is the reflexive and transitive *resource order*, $\mathcal{V}: M \rightarrow \text{Prop}$ is the *validity predicate*, $(\cdot): M \rightarrow M \rightarrow M$ is the *resource composition*, a commutative and associative binary operation on M , and $\varepsilon \in M$ is the *unit* of M , satisfying, for all $a, b, c \in M$:

$$\begin{array}{c} \mathcal{V}(\varepsilon) \\ \varepsilon \cdot a = a \end{array} \quad \frac{\mathcal{V}(a \cdot b)}{\mathcal{V}(a)} \quad \frac{a \preceq b \quad \mathcal{V}(b)}{\mathcal{V}(a)} \quad \frac{a \preceq b}{a \cdot c \preceq b \cdot c}$$

Any RA can serve as a model of basic connectives of separation logics; in particular, $P * Q$ will hold on $a \in M$ if and only if there are $a_1, a_2 \in M$ such that $a = a_1 \cdot a_2$ and P holds on a_1 and Q holds on a_2 .

BLUEBELL's assertions will be interpreted over a specific RA, which we construct as the combination of other basic RAs. The main component is the Probability Spaces RA, which uses independent product as the RA operation.

Definition 4.2 (Probability Spaces RA). The *probability spaces RA* PSp_Ω is the resource algebra $(\mathbb{P}(\Omega) \uplus \{\frac{1}{2}\}, \preceq, \mathcal{V}, \cdot, \mathbb{1}_\Omega)$ where \preceq is the extension order with $\frac{1}{2}$ added as the top element, that is, $\mathcal{P}_1 \preceq \mathcal{P}_2 \triangleq \mathcal{P}_1 \subseteq \mathcal{P}_2$ and $\forall a \in \text{PSp}_\Omega. a \preceq \frac{1}{2}$; $\mathcal{V}(a) \triangleq a \neq \frac{1}{2}$; composition is independent product:

$$a \cdot b \triangleq \begin{cases} \mathcal{P}_1 \otimes \mathcal{P}_2 & \text{if } a = \mathcal{P}_1, b = \mathcal{P}_2, \text{ and } \mathcal{P}_1 \otimes \mathcal{P}_2 \text{ is defined} \\ \frac{1}{2} & \text{otherwise} \end{cases}$$

The fact that PSp_Ω satisfies the axioms of RAs is established in Section D and builds on the analogous construction in Lilac. In comparison with the coarser model of PSL, independent product represents a more sophisticated way of separating probability spaces. In PSL, separation of distributions requires the distributions to involve disjoint sets of variables, ruling out assertions like $x \sim \mu * \llbracket x = y \rrbracket$ or $\text{own}(x) * \text{own}(x \text{ xor } y)$, which are satisfiable in Lilac and BLUEBELL.

Example 4.3. Assume there are only two variables x and y . Let $X_v = \{s \mid s(x) = v\}$ and $\mathcal{P}_1 = (\mathcal{F}_1, \mu_1)$ with $\mathcal{F}_1 = \sigma(\{X_v \mid v \in \mathbb{V}\})$ and let μ_1 give x the distribution of a fair coin, i.e. μ_1 is the extension to \mathcal{F}_1 of $\mu_1(X_0) = \mu_1(X_1) = \frac{1}{2}$. Intuitively, the assertion $x \sim \text{Ber}_{\frac{1}{2}}$ holds on \mathcal{P}_1 (we will define

the assertion's interpretation in Section 4.2). Similarly, $\lceil x = y \rceil$ holds on $\mathcal{P}_2 = (\mathcal{F}_2, \mu_2)$ where $\mathcal{F}_2 = \{\emptyset, \mathbb{S}, \{E\}, \mathbb{S} \setminus E\}$ with $E = \{s \mid s(x) = s(y)\}$ and $\mu_2(E) = 1$. Note that \mathcal{F}_2 is very coarse: it does not contain events that can pin the value of x precisely; thanks to this, μ_2 does not need to specify what is the distribution of x , but only that y will coincide on x with probability 1. It is easy to see that the independent product of \mathcal{P}_1 and \mathcal{P}_2 exists and is $\mathcal{P}_3 = (\mathcal{F}_1 \oplus \mathcal{F}_2, \mu_3)$ where μ_3 is determined by $\mu_3(X_0 \cap E) = \mu_3(X_1 \cap E) = \frac{1}{2}$, i.e. makes x y the outcomes of the same fair coin. This means \mathcal{P}_3 is a model of $x \sim \text{Ber}_{\frac{1}{2}} * \lceil x = y \rceil$.

When state is immutable, like in Lilac (which uses a functional language), the PSP_{Ω} RA is adequate to support a logic for probabilistic independence. There is however an obstacle in adopting independent product in a language with mutable state.

Example 4.4. Consider a simple assignment $x := 0$. In the spirit of separation logic's local reasoning, we would want to prove a small-footprint triple for the assignment, i.e. one where the precondition only involves ownership of the variable x . We could try with $x \sim \mu$ (for arbitrary μ) but we would run into problems proving the frame property: as we remarked, an assertion like $\lceil x = y \rceil$ is a valid frame of $x \sim \mu$; yet if $y \neq 0$, such frame would not hold after the assignment.

We solve this problem by combining PSP with an RA of permissions over variables. The idea is that in addition to information about the distribution, assertions can indicate which "write permissions" we own on variables. An assertion that owns write permissions on x would be incompatible with any frame predicating on x . Then a triple for assignment just needs to require write permission to the assigned variable. We model permissions using a standard fractional permission RA.

Definition 4.5. The *permissions RA* is defined as $(\text{Perm}, \preceq, \mathcal{V}, \cdot, \varepsilon)$ where $\text{Perm} \triangleq \mathbb{X} \rightarrow \mathbb{Q}^+$, $a \preceq b \triangleq \forall x \in \mathbb{X}. a(x) \leq b(x)$, $\mathcal{V}(a) \triangleq (\forall x \in \mathbb{X}. a(x) \leq 1)$, $a_1 \cdot a_2 \triangleq \lambda x. a_1(x) + a_2(x)$ and $\varepsilon = \lambda _. 0$.

We now want to combine permissions with probability spaces. The goal is to allow probability spaces to contain only information about variables of which we have some non-zero permission. This gives rise to the following definition.

Definition 4.6 (Compatibility). Given a probability space $\mathcal{P} \in \mathbb{P}(\mathbb{S})$ and a permission map $p \in \text{Perm}$, we say that \mathcal{P} is *compatible* with p , written $\mathcal{P} \# p$, if there exists $\mathcal{P}' \in \mathbb{P}((\mathbb{X} \setminus S) \rightarrow \mathbb{V})$ such that $\mathcal{P} = \mathcal{P}' \otimes \mathbb{1}_{S \rightarrow \mathbb{V}}$, where $S = \{x \in \mathbb{X} \mid p(x) = 0\}$. Note that we are exploiting the isomorphism $\mathbb{S} \cong ((\mathbb{X} \setminus S) \rightarrow \mathbb{V}) \times (S \rightarrow \mathbb{V})$. We extend the notion to $\text{PSP}_{\mathbb{S}}$ by declaring $\frac{1}{2} \# p \triangleq \text{True}$.

We can now construct an RA which combines probability spaces and permissions.

Definition 4.7. Let $\text{PSPm} \triangleq \{(\mathcal{P}_i, p) \mid \mathcal{P}_i \in \text{PSP}_{\mathbb{S}}, p \in \text{Perm}, \mathcal{P}_i \# p\}$. We associate with PSPm the *Probability Spaces with Permissions RA* ($\text{PSPm}, \preceq, \mathcal{V}, \cdot, \varepsilon$) where

$$\begin{aligned} \mathcal{V}((\mathcal{P}_i, p)) &\triangleq \mathcal{P}_i \neq \frac{1}{2} \wedge \forall x. p(x) \leq 1 & (\mathcal{P}_i, p) \cdot (\mathcal{P}'_i, p') &\triangleq (\mathcal{P}_i \cdot \mathcal{P}'_i, p \cdot p') \\ (\mathcal{P}_i, p) &\preceq (\mathcal{P}'_i, p') \triangleq \mathcal{P}_i \preceq \mathcal{P}'_i \text{ and } p \preceq p' & \varepsilon &\triangleq (\mathbb{1}_{\mathbb{S}}, \lambda x. 0) \end{aligned}$$

Example 4.8. Using PSPm , we can refine an assertion $x \sim \mu$ into $(x \sim \mu)@(x:q)$, which holds on resources (\mathcal{P}, p) where \mathcal{P} distributes x as μ and $p(x) \geq q$. What this achieves is to be able to differentiate between an assertion $(x \sim \mu)@(x:\frac{1}{2})$ which allows frames to predicate on x (e.g. $\lceil x = y \rceil$) and an assertion $(x \sim \mu)@(x:1)$ which does not allow the frame and consequently allows mutation of x .

While this allows for a clean semantic treatment of mutation and independence, it does incur some bookkeeping of permissions in practice, which we omitted in the examples of Section 2. The necessary permissions are however easy to infer from the variables used in the assertions, as we will illustrate later in Example 4.15.

Finally, to build BLUEBELL's model we simply construct an RA of I -indexed tuples of probability spaces with permissions.

Definition 4.9 (BLUEBELL RA). Given a set of indices I and a RA M , the *product* RA M^I is the pointwise lifting of the components of M . BLUEBELL's model is $\mathcal{M}_I \triangleq \text{PSPm}^I$.

4.2 Probabilistic Hyper-Assertions

Now we turn to the assertions in our logic. We take a semantic approach to assertions: we do not insist on a specific syntax and instead characterize what constitutes an assertion by its type. In Separation Logic, assertions are defined relative to some RA M , as the upward closed functions $M \rightarrow \text{Prop}$. An assertion $P: M \rightarrow \text{Prop}$ is upward closed if $\forall a, a' \in M. a \leq_M a' \Rightarrow P(a) \Rightarrow P(a')$. We write $M \xrightarrow{u} \text{Prop}$ for the type of upward closed assertions on M . We define hyper-assertions to be assertions over \mathcal{M}_I , i.e. $P \in \text{HA}_I \triangleq \mathcal{M}_I \xrightarrow{u} \text{Prop}$. Entailment is defined as $(P \vdash Q) \triangleq \forall a \in M. \mathcal{V}(a) \Rightarrow (P(a) \Rightarrow Q(a))$. Logical equivalence is defined as entailment in both directions: $P \Vdash Q \triangleq (P \vdash Q) \wedge (Q \vdash P)$. We inherit the basic connectives (conjunction, disjunction, separation, quantification) from SL, which are well-defined on arbitrary RAs, including \mathcal{M}_I . In particular:

$$P * Q \triangleq \lambda a. \exists b_1, b_2. (b_1 \cdot b_2) \leq a \wedge P(b_1) \wedge Q(b_2) \quad \ulcorner \phi \urcorner \triangleq \lambda _ . \phi \quad \text{Own}(b) \triangleq \lambda a. b \leq a$$

Pure assertions $\ulcorner \phi \urcorner$ lift meta-level propositions ϕ to assertions (by ignoring the resource). $\text{Own}(b)$ holds on resources that are greater or equal than b in the RA order; this means b represents a lower bound on the available resources.

We now turn to assertions that are specific to probabilistic reasoning in BLUEBELL, i.e. the ones that can only be interpreted in \mathcal{M}_I . We use the following two abbreviations:

$$\text{Own}(\mathcal{F}, \mu, p) \triangleq \text{Own}(((\mathcal{F}, \mu), p)) \quad \text{Own}(\mathcal{F}, \mu) \triangleq \exists p. \text{Own}(\mathcal{F}, \mu, p)$$

To start, we define *A-typed assertion expressions* E which are of type $E: \mathbb{S} \rightarrow A$. Note that the type of the semantics of a program expression $\llbracket e \rrbracket: \mathbb{S} \rightarrow \mathbb{V}$ is a \mathbb{V} -typed assertion expression; because of this we seamlessly use program expressions in assertions, implicitly coercing them to their semantics. Since in general we deal with hyper-stores $s \in \mathbb{S}^I$, we use the notation $E(i)$ to denote the application of E to the store $s(i)$. Notationally, it may be confusing to read composite expressions like $(x - z)(i)$, so we write them for clarity with each program variable annotated with the index, as in $x\langle i \rangle - z\langle i \rangle$.

The meaning of owning $x\langle 1 \rangle \sim \mu$. A function $f: A \rightarrow B$ is *measurable* in a σ -algebra $\mathcal{F}: \mathbb{A}(A)$ if $f^{-1}(b) = \{a \in A \mid f(a) = b\} \in \mathcal{F}$ for all $b \in B$. An expression E always defines a measurable function (i.e. a *random variable*) in $\Sigma_{\mathbb{S}}$, but might not be measurable in some sub-algebras of $\Sigma_{\mathbb{S}}$. Lilac proposed to use measurability as the notion of ownership: an expression E is owned in any resources that contains enough information to determine its distribution, i.e. that makes E measurable. While this makes sense conceptually, we discovered it made another important connective of Lilac, almost sure equality, slightly flawed (in that it would not support the necessary laws).² We propose a slight weakening of the notion of measurability which solves those issues while still retaining the intent behind the meaning of ownership in relation to independence and conditioning. We call this weaker notion “almost measurability”.

Definition 4.10 (Almost-measurability). Given a probability space $(\mathcal{F}, \mu) \in \mathbb{P}(\Omega)$ and a set $X \subseteq \Omega$, we say X is *almost measurable* in (\mathcal{F}, μ) , written $X \prec (\mathcal{F}, \mu)$, if

$$\exists X_1, X_2 \in \mathcal{F}. X_1 \subseteq X \subseteq X_2 \wedge \mu(X_1) = \mu(X_2)$$

²In fact, a later revision [Li et al. 2023b] corrected the issue, although with a different solution from ours. See Section 6.

We say a function $E: \Omega \rightarrow A$, is *almost measurable* in (\mathcal{F}, μ) , written $E \prec (\mathcal{F}, \mu)$, if $E^{-1}(a) \prec (\mathcal{F}, \mu)$ for all $a \in A$. When $X_1 \subseteq X \subseteq X_2$ and $\mu(X_1) = \mu(X_2) = p$, we can unambiguously assign probability p to X , as any extension of μ to Σ_Ω must assign p to X ; then we write $\mu(X)$ for p .

While almost-measurability does not imply measurability, it constrains the current probability space to contain enough information to uniquely determine the distribution of E in any extension where E becomes measurable. For example let $X = \{s \mid s(x) = 42\}$ and $\mathcal{F} = \sigma(\{X\}) = \{\emptyset, X, \mathbb{S} \setminus X, \mathbb{S}\}$. If $\mu(X) = 1$, then $x \prec (\mathcal{F}, \mu)$ holds but x is not measurable in \mathcal{F} , as \mathcal{F} lacks events for $x = v$ for all v except 42. Nevertheless, any extension $(\mathcal{F}', \mu') \supseteq (\mathcal{F}, \mu)$ where x is measurable, would need to assign $\mu'(X) = 1$ and $\mu(x = v) = 0$ for every $v \neq 42$.

We arrive at the definition of the assertion $E\langle i \rangle \sim \mu$ which requires $E\langle i \rangle$ to be almost-measurable, determining its distribution as μ in any extension of the local probability space. Formally, given $\mu: \mathbb{D}(\Sigma_A)$ and $E: \mathbb{S} \rightarrow A$, we define:

$$E\langle i \rangle \sim \mu \triangleq \exists \mathcal{F}, \mu. \text{Own}(\mathcal{F}, \mu) * \lceil E \prec (\mathcal{F}(i), \mu(i)) \wedge \mu = \mu(i) \circ E^{-1} \rceil$$

The assertion states that we own just enough information about the probability space at index i , so that its distribution is uniquely determined as μ in any extension of the space.

Using the $E\langle i \rangle \sim \mu$ assertion we can define a number of useful derived assertions:

$$\begin{aligned} E[E\langle i \rangle] = r &\triangleq \exists \mu. E\langle i \rangle \sim \mu * \lceil r = \sum_{a \in \text{supp}(\mu)} \mu(a) \cdot a \rceil & \lceil E\langle i \rangle \rceil &\triangleq (E \in \text{true})\langle i \rangle \sim \delta_{\text{True}} \\ \text{Pr}(E\langle i \rangle) = r &\triangleq \exists \mu. E\langle i \rangle \sim \mu * \lceil \mu(\text{true}) = r \rceil & \text{own}(E\langle i \rangle) &\triangleq \exists \mu. E\langle i \rangle \sim \mu \end{aligned}$$

Assertions about expectations ($E[E\langle i \rangle]$) and probabilities ($\text{Pr}(E\langle i \rangle)$), simply assert ownership of some distribution with the desired (pure) property. The “almost surely” assertion $\lceil E\langle i \rangle \rceil$ takes a boolean-valued expression E and asserts that it holds (at i) with probability 1. As remarked in Example 4.3, an assertion like $\lceil x\langle 1 \rangle = y\langle 1 \rangle \rceil$ owns the expression $(x\langle 1 \rangle = y\langle 1 \rangle)$ but not necessarily $x\langle 1 \rangle$ itself: the only events needed to make the expression almost measurable are $x\langle 1 \rangle = y\langle 1 \rangle$ and $x\langle 1 \rangle \neq y\langle 1 \rangle$, which would be not enough to make $x\langle 1 \rangle$ itself almost measurable. This means that an assertion like $\text{own}(x\langle 1 \rangle) * \lceil x\langle 1 \rangle = y\langle 1 \rangle \rceil$ is satisfiable.

Permissions. The previous example highlights the difficulty with supporting mutable state: owning $x\langle 1 \rangle \sim \mu$ is not enough to allow safe mutation, because the frame can record information like $\lceil x\langle 1 \rangle = y\langle 1 \rangle \rceil$, which could be invalidated by an assignment to x . Our solution is analogous to the “variables as resource” technique in Separation Logic [Bornat et al. 2005], and uses the permission component of BLUEBELL’s RA. To manipulate permissions we define the assertions:

$$(x\langle i \rangle : q) \triangleq \exists \mathcal{P}, p. \text{Own}(\mathcal{P}, p) * \lceil p(i)(x) = q \rceil \quad P@p \triangleq P \wedge \exists \mathcal{P}. \text{Own}(\mathcal{P}, p)$$

Now owning $(x\langle 1 \rangle : 1)$ forbids any frame to retain information about $x\langle 1 \rangle$: any resource compatible with $(x\langle 1 \rangle : 1)$ would have a σ -algebra which is trivial on $x\langle 1 \rangle$. In practice, preconditions are always of the form $P@p$ where p contains full permissions for every variable the relevant program mutates, and non-zero permissions for the other variables referenced in the assertions or program. When framing, one would distribute evenly the permissions to each separated conjunct, according to the variables mentioned in the assertions. We illustrate this pattern concretely in Example 4.15.

Relevant indices. Sometimes it is useful to determine which indices are relevant for an assertion. Semantically, we can determine if the indices $J \subseteq I$ are irrelevant to P by $\text{irrel}_J(P) \triangleq \forall a \in \mathcal{M}_I. (\exists a'. \mathcal{V}(a') \wedge a = a' \setminus J \wedge P(a')) \Rightarrow P(a)$. The set $\text{idx}(P)$ is the smallest subset of I so that $\text{irrel}_{I \setminus \text{idx}(P)}(P)$ holds. Section 4.2 states that separation between resources that live in different indexes is the same as normal conjunction: distributions at different indexes are neither independent nor correlated; they simply live in “parallel universes” and can be related as needed.

AND-TO-STAR

$$\text{idx}(P) \cap \text{idx}(Q) = \emptyset$$

$$\frac{P \wedge Q \vdash P * Q}{P \wedge Q \vdash P * Q}$$

$$\begin{array}{c}
\text{C-TRUE} \\
\vdash C_\mu \text{. True}
\end{array}
\qquad
\begin{array}{c}
\text{C-UNIT-L} \\
C_{\delta_{v_0}} v. K(v) \dashv\vdash K(v_0)
\end{array}
\qquad
\begin{array}{c}
\text{C-TRANSF} \\
\frac{f: \text{supp}(\mu') \rightarrow \text{supp}(\mu) \text{ bijective} \quad \forall b \in \text{supp}(\mu'). \mu'(b) = \mu(f(b))}{C_\mu a. K(a) \vdash C_{\mu'} b. K(f(b))}
\end{array}$$

$$\begin{array}{c}
\text{C-AND} \\
\frac{\text{idx}(K_1) \cap \text{idx}(K_2) = \emptyset}{C_\mu v. K_1(v) \wedge C_\mu v. K_2(v) \vdash C_\mu v. (K_1(v) \wedge K_2(v))}
\end{array}
\qquad
\begin{array}{c}
\text{SURE-STR-CONVEX} \\
C_\mu v. (K(v) * \lceil E\langle i \rangle \rceil) \vdash \lceil E\langle i \rangle \rceil * C_\mu v. K(v)
\end{array}$$

$$\begin{array}{c}
\text{C-PURE} \\
\lceil \mu(X) = 1^\top * C_\mu v. K(v) \dashv\vdash C_\mu v. (\lceil v \in X^\top * K(v) \rceil)
\end{array}$$

Fig. 4. Primitive Conditioning Laws.

4.3 Joint Conditioning

As we discussed in Section 2, the centerpiece of BLUEBELL is the joint conditioning modality, which we can now define fully formally.

Definition 4.11 (Joint conditioning modality). Let $\mu \in \mathbb{D}(\Sigma_A)$ and $K: A \rightarrow \text{HA}_I$, then we define the assertion $C_\mu K: \text{HA}_I$ as follows (where $\kappa(I)(v) \triangleq [i: \kappa(i)(v) \mid i \in I]$):

$$\begin{aligned}
C_\mu K \triangleq \lambda a. \exists \mathcal{F}, \mu, \mathbf{p}, \kappa. (\mathcal{F}, \mu, \mathbf{p}) \preceq a \wedge \forall i \in I. \mu(i) = \text{bind}(\mu, \kappa(i)) \\
\wedge \forall v \in \text{supp}(\mu). K(v)(\mathcal{F}, \kappa(I)(v), \mathbf{p})
\end{aligned}$$

The definition follows the principle we explained in Section 2.2: $C_\mu K$ holds on resources where we own some tuple of probability spaces which can all be seen as the convex combinations of the same μ and some kernel. Then the conditional assertion $K(v)$ is required to hold on the tuple of kernels evaluated at v . Note that the definition is upward-closed by construction.

We discussed a number of joint conditioning laws in Section 2. Figure 4 shows some important primitive laws that were left out. Section 4.3 allows to introduce a trivial modality; together with C-FRAME this allows for the introduction of the modality around any assertion. Section 4.3 is a reflection of the left unit rule of the underlying monad: conditioning on the Dirac distribution can be eliminated. Section 4.3 allows for the transformation of the convex combination using μ into using μ' by applying a bijection between their support in a way that does not affect the weights of each outcome. Section 4.3 allows to merge two modalities using the same μ , provided the inner conditioned assertions do not overlap in their relevant indices.

Section 4.3 internalizes a stronger version of convexity of $\lceil E\langle i \rangle \rceil$ assertions. When $K(v) = \text{True}$ we obtain convexity $C_\mu v. \lceil E\langle i \rangle \rceil \vdash \lceil E\langle i \rangle \rceil$. Additionally the rule asserts that the unconditional $\lceil E\langle i \rangle \rceil$ keeps being independent of the conditional K .

Finally, Section 4.3 allows to translate facts that hold with probability 1 in μ to predicates that hold on every v bound by conditioning on μ .

We can now give the general encoding of relational lifting in terms of joint conditioning.

Definition 4.12 (Relational Lifting). Let $X \subseteq I \times \mathbb{X}$; given a relation R between variables in X , i.e. $R \subseteq \mathbb{V}^X$, we define (letting $\lceil x\langle i \rangle = v\langle x\langle i \rangle \rangle \rceil_{x\langle i \rangle \in X} \triangleq \bigwedge_{x\langle i \rangle \in X} \lceil x\langle i \rangle = v\langle x\langle i \rangle \rangle \rceil$):

$$\llbracket R \rrbracket \triangleq \exists \mu: \mathbb{D}(\mathbb{V}^X). \lceil \mu(R) = 1^\top * C_\mu v. \lceil x\langle i \rangle = v\langle x\langle i \rangle \rangle \rceil_{x\langle i \rangle \in X}$$

Example 4.13. Let us expand Definition 4.12 on $\llbracket k\langle 1 \rangle = c\langle 2 \rangle \rrbracket$. The assertion concerns the variables $X = \{k\langle 1 \rangle, c\langle 2 \rangle\}$; to instantiate the definition we see the assertion as the lifting $\llbracket R_- \rrbracket$ of

the relation $R_{=} \subseteq \mathbb{V}^X$ defined as $R_{=} = \{v \in \mathbb{V}^X \mid v(k\langle 1 \rangle) = v(c\langle 2 \rangle)\}$, giving rise to the assertion

$$\exists \mu. \ulcorner \mu(R_{=}) = 1 \urcorner * C_{\mu} v. [\![k\langle 1 \rangle = v(k\langle 1 \rangle)]\!] \wedge [\![c\langle 2 \rangle = v(c\langle 2 \rangle)]\!]$$

Here, \mathbb{V}^X can be alternatively presented as \mathbb{V}^2 , giving $R_{=} \equiv \{(v, v) \mid v \in \mathbb{V}\}$. With this reformulation, the encoding of Definition 4.12 becomes

$$\exists \mu. \ulcorner \mu(R_{=}) = 1 \urcorner * C_{\mu}(v_1, v_2). [\![k\langle 1 \rangle = v_1]\!] \wedge [\![c\langle 2 \rangle = v_2]\!]$$

Thanks to **C-PURE**, the assertion can be rewritten as $\exists \mu. C_{\mu}(v_1, v_2). \ulcorner R_{=}(v_1, v_2) \urcorner * [\![k\langle 1 \rangle = v_1]\!] \wedge [\![c\langle 2 \rangle = v_2]\!]$ which can be simplified to $\exists \mu. C_{\mu}(v_1, v_2). ([\![k\langle 1 \rangle = v_1]\!] \wedge [\![c\langle 2 \rangle = v_2]\!] \wedge \ulcorner v_1 = v_2 \urcorner)$ (which is how we presented the encoding in Section 2.2). Since $R_{=}$ is so simple, by **C-TRANSF** we can simplify the assertion even further and obtain $\exists \mu. C_{\mu} v. ([\![k\langle 1 \rangle = v]\!] \wedge [\![c\langle 2 \rangle = v]\!])$.

In Section 2.4, the two relations might refer to different indexed variables, i.e. $R_1 \in \mathbb{V}^{X_1}$ and $R_2 \in \mathbb{V}^{X_2}$; the notation $R_1 \wedge R_2$ is defined as $R_1 \wedge R_2 \triangleq \{s \in \mathbb{V}^{X_1 \cup X_2} \mid s|_{X_1} \in R_1 \wedge s|_{X_2} \in R_2\}$.

4.4 Weakest Precondition

To reason about (hyper-)programs, we introduce a *weakest-precondition assertion* (WP) $\mathbf{wp} \, t \, \{Q\}$, which intuitively states: given the current input distributions (at each index), if we run the programs in t at their corresponding index we obtain output distributions that satisfy Q ; furthermore, every frame is preserved. We refer to the number of indices of t as the *arity* of the WP.

Definition 4.14 (Weakest Precondition). For $a \in \mathcal{M}_I$ and $\mu : \mathbb{D}(\Sigma_{\mathbb{S}^I})$ let $a \preceq \mu$ mean $a \preceq (\Sigma_{\mathbb{S}^I}, \mu, \lambda x. 1)$.

$$\mathbf{wp} \, t \, \{Q\} \triangleq \lambda a. \forall \mu_0. \forall c. (a \cdot c) \preceq \mu_0 \Rightarrow \exists b. ((b \cdot c) \preceq \llbracket t \rrbracket(\mu_0) \wedge Q(b))$$

The assertion holds on the resources a such that if, together with some frame c , they can be seen as a fragment of the global distribution μ_0 , then it is possible to update the resource to some b which still composes with the frame c , and $b \cdot c$ can be seen as a fragment of the output distribution $\llbracket t \rrbracket(\mu_0)$. Moreover, such b needs to satisfy the postcondition Q .

We discussed some of the WP rules of BLUEBELL in Section 2; the full set of rules is produced in Section A. Let us briefly mention the axioms for assignments:

<p>WP-SAMP</p> $(x\langle i \rangle : 1) \vdash \mathbf{wp} \, [i : x \approx d(\vec{v})] \, \{x\langle i \rangle \sim d(\vec{v})\}$	<p>WP-ASSIGN</p> $\frac{x \notin \text{pvar}(e) \quad \forall y \in \text{pvar}(e). \mathbf{p}(y\langle i \rangle) > 0 \quad \mathbf{p}(x\langle i \rangle) = 1}{(\mathbf{p}) \vdash \mathbf{wp} \, [x := e] \, \{[x\langle i \rangle = e\langle i \rangle] @ \mathbf{p}\}}$
---	---

Section 4.4 is the expected “small footprint” rule for sampling; the precondition only requires full permission on the variable being assigned, to forbid any frame to record information about it. Section 4.4 requires full permission on x , and non-zero permission on the variables on the RHS of the assignment. This allows the postcondition to assert that x and the expression e assigned to it are equal with probability 1. The condition $x \notin \text{pvar}(\vec{e})$ ensures e has the same meaning before and after the assignment, but is not restrictive: if needed the old value of x can be stored in a temporary variable, or the proof can condition on x to work with its pure value.

The assignment rules are the only ones that impose constraints on the owned permissions. In proofs, this means that most manipulations simply thread through permissions so that the needed ones can reach the applications of the assignment rules. To avoid cluttering derivations with this bookkeeping, we mostly omit permission information from assertions. The appropriate permission annotations can be easily inferred, as we show in the following example.

Example 4.15. Consider the following triple with permissions omitted:

$$x\langle 1 \rangle \sim \mu_1 * [\![x\langle 1 \rangle = y\langle 1 \rangle]\!] * z\langle 1 \rangle \sim \mu_2 \vdash \mathbf{wp} \, [1 : x := z] \, \{[\![x\langle 1 \rangle = z\langle 1 \rangle]\!] * z\langle 1 \rangle \sim \mu_2\}$$

To be able to apply Section 4.4, we need to get $(x:1, z:q)$ from the precondition, for some $q > 0$. To do so, formally, we need to be more explicit about the permissions owned. The pattern is that whenever a triple is considered, the precondition should own full permissions on the variables assigned to in the program term, and non-zero permission on the other relevant variables. In our example we need permission 1 for $x\langle 1 \rangle$ and arbitrary permissions $q_1, q_2 > 0$ for $y\langle 1 \rangle$ and $z\langle 1 \rangle$ respectively. Since we have two separated sub-assertions that refer to $x\langle 1 \rangle$, we would split the full permission into two halves, obtaining the precondition:

$$(x\langle 1 \rangle \sim \mu_1) @ (x\langle 1 \rangle : \frac{1}{2}) * [x\langle 1 \rangle = y\langle 1 \rangle] @ (x\langle 1 \rangle : \frac{1}{2}, y\langle 1 \rangle : q_1) * (z\langle 1 \rangle \sim \mu_2) @ (z\langle 1 \rangle : q_2)$$

To obtain the full permission on $x\langle 1 \rangle$ we are now forced to consume both the first two resources, weakening the precondition to:

$$(x\langle 1 \rangle : 1) * (y\langle 1 \rangle : q_1) * (z\langle 1 \rangle \sim \mu_2) @ (z\langle 1 \rangle : q_2)$$

This step in general forces the consumption of any frame recording information about the assigned variables. To obtain non-zero permission for $z\langle 1 \rangle$ while still being able to frame $z\langle 1 \rangle \sim \mu_2$, we let $q = q_2/2$ and weaken the precondition to:

$$(x\langle 1 \rangle : 1, z\langle 1 \rangle : q) * (y\langle 1 \rangle : q_1) * (z\langle 1 \rangle \sim \mu_2) @ (z\langle 1 \rangle : q)$$

Now an application of **WP-FRAME** and **WP-ASSIGN** would give us a postcondition:

$$[x\langle 1 \rangle = z\langle 1 \rangle] @ (x\langle 1 \rangle : 1, z\langle 1 \rangle : q) * (y\langle 1 \rangle : q_1) * (z\langle 1 \rangle \sim \mu_2) @ (z\langle 1 \rangle : q)$$

which is strong enough to imply the desired postcondition. In a fully expanded proof, one would keep the permissions owned in the postcondition so that they can be used in proofs concerning the continuation of the program.

5 Case Studies for BLUEBELL

Our evaluation of BLUEBELL is based on two main lines of enquiry: (1) Are high-level principles about probabilistic reasoning provable from the core constructs of BLUEBELL? (2) Does BLUEBELL, through enabling new reasoning patterns, expand the horizon for verification of probabilistic programs *beyond* what was possible before? We include case studies that try to highlight the contribution of BLUEBELL each question, and sometimes both at the same time. Specifically, our evaluation is guided by the following research questions:

- RQ1:** Do joint conditioning and independence offer a good abstract interface over the underlying semantic model?
- RQ2:** Can known unary/relational principles be reconstructed from BLUEBELL's primitives?
- RQ3:** Can new unary/relational principles be discovered (as new lemmas) and proved from BLUEBELL's primitives?
- RQ4:** Can BLUEBELL's primitives be successfully incorporated in an effective *program* logic?

We already demonstrated positive answers to some of these questions in Section 2: for example, the proof of the One-time pad addresses **RQ1** and **RQ2**, the proof of **SEQ-SWAP** addresses **RQ3** and **RQ4**. In this section we provide a more detailed evaluation through a number of challenging examples. The full proofs of the case studies and additional examples are in Section G. Here, we summarize some highlights to frame the key contributions of BLUEBELL.

5.1 pRHL-style Reasoning

Our first example is an encoding of pRHL's judgments in BLUEBELL, sketching how pRHL-style reasoning can be effectively embedded and extended in BLUEBELL (**RQ1** to **RQ4**).

In pRHL, the semantics of triples implicitly always conditions on the input store, so that programs are always seen as running from a pair of *deterministic* input stores satisfying the relational precondition. Judgments in pRHL have the form $\vdash t_1 \sim t_2 : R_0 \Rightarrow R_1$ where R_0, R_1 are two relations on states (the pre- and postcondition, respectively) and t_1, t_2 are the two programs to be compared. Such a judgment can be encoded in BLUEBELL as:

$$\llbracket R_0 \rrbracket \vdash \exists \mu. C_\mu s. (St(s) \wedge \mathbf{wp} [1: t_1, 2: t_2] \{ \llbracket R_1 \rrbracket \}) \quad \text{where} \quad St(s) \triangleq \llbracket x\langle i \rangle = s(x\langle i \rangle) \rrbracket_{x\langle i \rangle \in I \times \mathbb{X}} \quad (7)$$

As the input state is always conditioned, and the precondition is always a relational lifting, one is always in the position of applying **C-CONS** to eliminate the implicit conditioning of the lifting and the one wrapping the WP, reducing the problem to a goal where the input state is deterministic (and thus where the primitive rules of WP laws apply without need for further conditioning). As noted in Section 2.4, LHC-style WPs allow us to lift our unary WP rules to binary with little effort.

An interesting property of the encoding in (7) is that anything of the form $C_\mu s. (St(s) \wedge \dots)$ has ownership of the full store (as it conditions on every variable). We observe that WPs (of any arity) which have this property enjoy an extremely powerful rule. Let $\text{own}_{\mathbb{X}} \triangleq \forall x\langle i \rangle \in I \times \mathbb{X}. \text{own}(x\langle i \rangle)$. The following is a valid (primitive) rule in BLUEBELL:

C-WP-SWAP

$$C_\mu v. \mathbf{wp} t \{Q(v)\} \wedge \text{own}_{\mathbb{X}} \vdash \mathbf{wp} t \{C_\mu v. Q(v)\}$$

Section 5.1, allows the shift of the conditioning on the input to the conditioning of the output. This rule provides a powerful way to make progress in lifting a conditional statement to an unconditional one. To showcase **C-WP-SWAP**, consider the two programs in Fig. 7, which are equivalent: if we couple the x in both programs, the other two samplings can be coupled under conditioning on x . Formally, let $P \Vdash Q \triangleq P \wedge \text{own}_{\mathbb{X}} \vdash Q \wedge \text{own}_{\mathbb{X}}$. We process the two assignments to x , which we can couple $x\langle 1 \rangle \sim d_0 * x\langle 2 \rangle \sim d_0 \vdash C_{d_0} v. (\llbracket x\langle 1 \rangle = v \rrbracket \wedge \llbracket x\langle 2 \rangle = v \rrbracket)$. Then, let t_1 (t_2) be the rest of prog1 (prog2). We can then derive:

$$\begin{array}{c} \frac{\forall v. \llbracket x\langle 1 \rangle = v \rrbracket \wedge \llbracket x\langle 2 \rangle = v \rrbracket \Vdash \mathbf{wp} [1: t_1, 2: t_2] \left\{ \begin{array}{l} \llbracket x\langle 1 \rangle = x\langle 2 \rangle \rrbracket * y\langle 1 \rangle \sim d_1(v) * y\langle 2 \rangle \sim d_1(v) * \\ \llbracket z\langle 1 \rangle \sim d_2(v) * z\langle 2 \rangle \sim d_2(v) \rrbracket \end{array} \right\}}{\forall v. \llbracket x\langle 1 \rangle = v \rrbracket \wedge \llbracket x\langle 2 \rangle = v \rrbracket \Vdash \mathbf{wp} [1: t_1, 2: t_2] \{ \llbracket x\langle 1 \rangle = x\langle 2 \rangle \rrbracket * \llbracket y\langle 1 \rangle = y\langle 2 \rangle \rrbracket * \llbracket z\langle 1 \rangle = z\langle 2 \rangle \rrbracket \}} \text{COUPLING} \\ \frac{\forall v. \llbracket x\langle 1 \rangle = v \rrbracket \wedge \llbracket x\langle 2 \rangle = v \rrbracket \Vdash \mathbf{wp} [1: t_1, 2: t_2] \{ \llbracket x\langle 1 \rangle = x\langle 2 \rangle \rrbracket \wedge y\langle 1 \rangle = y\langle 2 \rangle \wedge z\langle 1 \rangle = z\langle 2 \rangle \rrbracket }}{\forall v. \llbracket x\langle 1 \rangle = v \rrbracket \wedge \llbracket x\langle 2 \rangle = v \rrbracket \Vdash \mathbf{wp} [1: t_1, 2: t_2] \{ \llbracket x\langle 1 \rangle = x\langle 2 \rangle \rrbracket \wedge y\langle 1 \rangle = y\langle 2 \rangle \wedge z\langle 1 \rangle = z\langle 2 \rangle \rrbracket \}} \text{RL-MERGE} \\ \frac{C_{d_0} v. (\llbracket x\langle 1 \rangle = v \rrbracket \wedge \llbracket x\langle 2 \rangle = v \rrbracket) \Vdash C_{d_0} v. \mathbf{wp} [1: t_1, 2: t_2] \{ \llbracket x\langle 1 \rangle = x\langle 2 \rangle \rrbracket \wedge y\langle 1 \rangle = y\langle 2 \rangle \wedge z\langle 1 \rangle = z\langle 2 \rangle \rrbracket \}}{C_{d_0} v. (\llbracket x\langle 1 \rangle = v \rrbracket \wedge \llbracket x\langle 2 \rangle = v \rrbracket) \Vdash \mathbf{wp} [1: t_1, 2: t_2] \{ C_{d_0} v. \llbracket x\langle 1 \rangle = x\langle 2 \rangle \rrbracket \wedge y\langle 1 \rangle = y\langle 2 \rangle \wedge z\langle 1 \rangle = z\langle 2 \rangle \rrbracket \}} \text{C-CONS} \\ \frac{C_{d_0} v. (\llbracket x\langle 1 \rangle = v \rrbracket \wedge \llbracket x\langle 2 \rangle = v \rrbracket) \Vdash \mathbf{wp} [1: t_1, 2: t_2] \{ C_{d_0} v. \llbracket x\langle 1 \rangle = x\langle 2 \rangle \rrbracket \wedge y\langle 1 \rangle = y\langle 2 \rangle \wedge z\langle 1 \rangle = z\langle 2 \rangle \rrbracket \}}{C_{d_0} v. (\llbracket x\langle 1 \rangle = v \rrbracket \wedge \llbracket x\langle 2 \rangle = v \rrbracket) \Vdash \mathbf{wp} [1: t_1, 2: t_2] \{ \llbracket x\langle 1 \rangle = x\langle 2 \rangle \rrbracket \wedge y\langle 1 \rangle = y\langle 2 \rangle \wedge z\langle 1 \rangle = z\langle 2 \rangle \rrbracket \}} \text{C-WP-SWAP} \\ \frac{C_{d_0} v. (\llbracket x\langle 1 \rangle = v \rrbracket \wedge \llbracket x\langle 2 \rangle = v \rrbracket) \Vdash \mathbf{wp} [1: t_1, 2: t_2] \{ \llbracket x\langle 1 \rangle = x\langle 2 \rangle \rrbracket \wedge y\langle 1 \rangle = y\langle 2 \rangle \wedge z\langle 1 \rangle = z\langle 2 \rangle \rrbracket \}}{C_{d_0} v. (\llbracket x\langle 1 \rangle = v \rrbracket \wedge \llbracket x\langle 2 \rangle = v \rrbracket) \Vdash \mathbf{wp} [1: t_1, 2: t_2] \{ \llbracket x\langle 1 \rangle = x\langle 2 \rangle \rrbracket \wedge y\langle 1 \rangle = y\langle 2 \rangle \wedge z\langle 1 \rangle = z\langle 2 \rangle \rrbracket \}} \text{RL-CONVEX} \end{array}$$

Where the top triple can be easily derived using standard steps. Reading it from bottom to top, we start by invoking convexity of relational lifting to introduce a conditioning modality in the postcondition matching the one in the precondition. Section 5.1 allows us to bring the whole WP under the modality, allowing Section 2.3 to remove it on both sides. From then it is a matter of establishing and combining the couplings on y and z . Note that these couplings are only possible because the coupling on x made the parameters of d_1 and of d_2 coincide on both indices. In Section 5.5 we show this kind of derivation can be useful for unary reasoning too.

While the $\text{own}_{\mathbb{X}}$ condition is restricting, without it the rule is unsound in the current model. We leave it as future work to study whether there is a model that validates this rule without requiring $\text{own}_{\mathbb{X}}$.

5.2 One Time Pad Revisited

In Section 2, we prove the `encrypt` program correct relationally (missing details are in Section G.2). An alternative way of stating and proving the correctness of `encrypt` is to establish that in the output distribution c and m are independent, which can be expressed as the *unary* goal (also studied in [Barthe et al. 2019]): $(p) \vdash \mathbf{wp} [1:\text{encrypt}()] \{c\langle 1 \rangle \sim \text{Ber}_{1/2} * m\langle 1 \rangle \sim \text{Ber}_p\}$ (where $p = [k\langle 1 \rangle: 1, m\langle 1 \rangle: 1, c\langle 1 \rangle: 1]$). The triple states that after running `encrypt`, the ciphertext c is distributed as a fair coin, and—importantly—is *not* correlated with the plaintext in m . The PSL proof in [Barthe et al. 2019] performs some of the steps within the logic, but needs to carry out some crucial entailments at the meta-level, which is a symptom of unsatisfactory abstractions (RQ1). The same applies to the Lilac proof in [Li et al. 2023b] which requires ad-hoc lemmas proven on the semantic model. The stumbling block is proving the valid entailment:

$$k\langle 1 \rangle \sim \text{Ber}_{1/2} * m\langle 1 \rangle \sim \text{Ber}_p * [c\langle 1 \rangle = k\langle 1 \rangle \text{ xor } m\langle 1 \rangle] \vdash m\langle 1 \rangle \sim \text{Ber}_p * c\langle 1 \rangle \sim \text{Ber}_{1/2}$$

In BLUEBELL we can prove the entailment in two steps: (1) we condition on m and k to compute the result of the xor operation and obtain that c is distributed as $\text{Ber}_{1/2}$; (2) we carefully eliminate the conditioning while preserving the independence of m and c .

The first step starts by conditioning on m and k and proceeds as follows:

$$\begin{aligned} & C_{\text{Ber}_p} m. ([m\langle 1 \rangle = m] * C_{\text{Ber}_{1/2}} k. ([k\langle 1 \rangle = k] * [c\langle 1 \rangle = k \text{ xor } m])) \\ \vdash & C_{\text{Ber}_p} m. \left([m\langle 1 \rangle = m] * \begin{cases} C_{\text{Ber}_{1/2}} k. [c\langle 1 \rangle = k] & \text{if } m = 0 \\ C_{\text{Ber}_{1/2}} k. [c\langle 1 \rangle = \neg k] & \text{if } m = 1 \end{cases} \right) \quad (\text{C-CONS}) \\ \vdash & C_{\text{Ber}_p} m. ([m\langle 1 \rangle = m] * C_{\text{Ber}_{1/2}} k. [c\langle 1 \rangle = k]) \quad (\text{C-TRANSF}) \end{aligned}$$

The crucial entailment is the application of **C-TRANSF** to the $m = 1$ branch, by using negation as the bijection (which satisfies the premises of the rules since $\text{Ber}_{1/2}$ is unbiased).

The second step uses the following primitive rule of BLUEBELL:

$$\begin{aligned} & \text{PROD-SPLIT} \\ & (E_1\langle i \rangle, E_2\langle i \rangle) \sim \mu_1 \otimes \mu_2 \vdash E_1\langle i \rangle \sim \mu_1 * E_2\langle i \rangle \sim \mu_2 \end{aligned}$$

with which we can prove:

$$\begin{aligned} & C_{\text{Ber}_p} m. ([m\langle 1 \rangle = m] * C_{\text{Ber}_{1/2}} k. [c\langle 1 \rangle = k]) \\ \vdash & C_{\text{Ber}_p} m. C_{\text{Ber}_{1/2}} k. [m\langle 1 \rangle = m \wedge c\langle 1 \rangle = k] \quad (\text{C-FRAME, SURE-MERGE}) \\ \vdash & C_{\text{Ber}_p \otimes \text{Ber}_{1/2}} (m, k). [m\langle 1 \rangle, c\langle 1 \rangle] = (m, k) \quad (\text{C-FUSE}) \\ \vdash & (m\langle 1 \rangle, c\langle 1 \rangle) \sim (\text{Ber}_p \otimes \text{Ber}_{1/2}) \quad (\text{C-UNIT-R}) \\ \vdash & m\langle 1 \rangle \sim \text{Ber}_p * c\langle 1 \rangle \sim \text{Ber}_{1/2} \quad (\text{PROD-SPLIT}) \end{aligned}$$

As this is a common manipulation needed to extract unconditional independence from a conditional fact, we can formulate it as the more general derived rule

$$\begin{aligned} & \text{C-EXTRACT} \\ & C_{\mu_1} v_1. ([E_1\langle i \rangle = v_1] * E_2\langle i \rangle \sim \mu_2) \vdash E_1\langle i \rangle \sim \mu_1 * E_2\langle i \rangle \sim \mu_2 \end{aligned}$$

5.3 Markov Blankets

In probabilistic reasoning, introducing conditioning is easy, but deducing unconditional facts from conditional ones is not immediate. The same applies to the joint conditioning modality: by design, one cannot eliminate it for free. Crucial to BLUEBELL's expressiveness is the inclusion of rules that can soundly derive unconditional information from conditional assertions.

We use the concept of a *Markov Blanket*—a very common tool in Bayesian reasoning for simplifying conditioning—to illustrate BLUEBELL’s expressiveness (**RQ1** and **RQ2**). Intuitively, Markov blankets identify a set of variables that affect the distribution of a random variable *directly*: this is useful because by conditioning on those variables we can remove conditional connection between the random variable and all the variables on which it *indirectly* depends.

For concreteness, consider the program $x_1 \approx d_1$; $x_2 \approx d_2(x_1)$; $x_3 \approx d_3(x_2)$. The program describes a Markov chain of three variables. One way of interpreting this pattern is that the joint output distribution is described by the program as a product of conditional distributions: the distribution over x_2 is described conditionally on x_1 , and the one of x_3 conditionally on x_2 . This kind of dependencies are ubiquitous in, for instance, hidden Markov models and Bayesian network representations of distributions.

A crucial tool for the analysis of such models is the concept of a Markov Blanket of a variable x : the set of variables that are direct dependencies of x . Clearly x_3 depends on x_2 and, indirectly, on x_1 . However, Markov chains enjoy the memorylessness property: when fixing a variable in the chain, the variables that follow it are independent from the variables that preceded it. For our example this means that if we condition on x_2 , then x_1 and x_3 are independent (i.e. we can ignore the indirect dependencies).

In BLUEBELL we can characterize the output distribution with the assertion

$$C_{d_1} v_1. \left(\lceil x_1 = v_1 \rceil * C_{d_2(v_1)} v_2. (\lceil x_2 = v_2 \rceil * x_3 \sim d_3(v_2)) \right)$$

Note how this postcondition represents the output distribution as implicitly as the program does. We want to transform the assertion into:

$$C_{\mu_2} v_2. (\lceil x_2 = v_2 \rceil * x_1 \sim \mu_1(v_2) * x_3 \sim d_3(v_2))$$

for appropriate μ_2 and μ_1 . This isolates the conditioning to the direct dependency of x_1 and keeps full information about x_3 , available for further manipulation down the line.

In probability theory, the proof of memorylessness is an application of Bayes’ law: we are computing the distribution of x_1 conditioned on x_2 , from the distribution of x_2 conditioned on x_1 .

In BLUEBELL we can produce the transformation using the joint conditioning rules, in particular the right-to-left direction of **C-FUSE** and the primitive rule that is behind its left-to-right direction:

C-UNASSOC

$$C_{\text{bind}(\mu, \kappa)} w. K(w) \vdash C_\mu v. C_{\kappa(v)} w. K(w)$$

Using these we can prove:

$$\begin{aligned} & C_{d_1} v_1. \left(\lceil x_1 = v_1 \rceil * C_{d_2(v_1)} v_2. (\lceil x_1 = v_2 \rceil * x_3 \sim d_3(v_2)) \right) \\ \vdash & C_{d_1} v_1. \left(C_{d_2(v_1)} v_2. (\lceil x_1 = v_1 \rceil * \lceil x_1 = v_2 \rceil * x_3 \sim d_3(v_2)) \right) & \text{(C-FRAME)} \\ \vdash & C_{\mu_0}(v_1, v_2). (\lceil x_1 = v_1 \rceil * \lceil x_2 = v_2 \rceil * x_3 \sim d_3(v_2)) & \text{(C-FUSE)} \\ \vdash & C_{\mu_2} v_2. \left(C_{\mu_1(v_2)} v_1. (\lceil x_1 = v_1 \rceil * \lceil x_2 = v_2 \rceil * x_3 \sim d_3(v_2)) \right) & \text{(C-UNASSOC)} \\ \vdash & C_{\mu_2} v_2. \left(\lceil x_2 = v_2 \rceil * C_{\mu_1(v_2)} v_1. (\lceil x_1 = v_1 \rceil * x_3 \sim d_3(v_2)) \right) & \text{(SURE-STR-CONVEX)} \\ \vdash & C_{\mu_2} v_2. (\lceil x_2 = v_2 \rceil * x_1 \sim \mu_1(v_2) * x_3 \sim d_3(v_2)) & \text{(C-EXTRACT)} \end{aligned}$$

where $d_1 \prec d_2 = \mu_0 = \text{bind}(\mu_2, \mu_1)$. The existence of such μ_2 and μ_1 is a simple application of Bayes’ law: $\mu_2(v_2) = \sum_{v_1 \in \mathbb{V}} \mu_0(v_1, v_2)$, and $\mu_1(v_2)(v_1) = \frac{\mu_0(v_1, v_2)}{\mu_2(v_2)}$. We see the ability of BLUEBELL to perform these manipulations as evidence that joint conditioning and independence form a sturdy

abstraction over the semantic model (RQ1). The amount of meta-reasoning required to manipulate the distributions indexing the conditioning modality are minimal and localized, and offer a good entry-point to inject facts about distributions without interfering with the rest of the proof context.

5.4 Multi-party Secure Computation

In *multi-party secure computation*, the goal is to for N parties to compute a function $f(x_1, \dots, x_N)$ of some private data x_i owned by each party i , without revealing any more information about x_i than the output of f would reveal if computed centrally by a trusted party. For example, if f is addition, a secure computation of f can be used to compute the total number of votes without revealing who voted positively: some information would leak (e.g. if the total is non-zero then *somebody* voted positively) but only what is revealed by knowing the total and nothing more.

To achieve this objective, multi-party secure addition (MPSAdd) works by having the parties break their secret into N *secret shares* which individually look random, but the sum of which amounts to the original secret. These secret shares are then distributed to the other parties so that each party knows an incomplete set of shares of the other parties. Yet, each party can reliably compute the result of the function by computing a function of the received shares.

As it is very often the case, there is no single “canonical” way of specifying this kind of security property. For MPSAdd, for instance, we can formalize security (focusing on the perspective of party 1) in two ways: as a unary or as a relational specification.

The *unary specification* says that, conditionally on the secret of party 1 and the sum of the other secrets, all the values received by 1 (we call this the *view* of 1) are independent from the secrets of the other parties. Roughly:

$$(x_1, x_2, x_3)\langle 1 \rangle \sim \mu_0 \vdash \mathbf{wp} [1: \text{MPSAdd}] \left\{ \exists \mu. C_\mu(v, s). \left(\left[x_1\langle 1 \rangle = v \wedge (x_2 + x_3)\langle 1 \rangle = s \right] * \left[\text{own}(\text{view}_1\langle 1 \rangle) * \text{own}(x_2\langle 1 \rangle, x_3\langle 1 \rangle) \right] \right) \right\}$$

where μ_0 is an arbitrary distribution of the three secrets. Notice how conditioning nicely expresses that the acceptable leakage is just the sum.

The *relational specification* says that when running the program from two initial states differing only in the secrets of the other parties, but not in their sum, the views of party i would be distributed in the same way. Roughly:

$$\left[\begin{array}{l} x_1\langle 1 \rangle = x_1\langle 2 \rangle \\ (x_2 + x_3)\langle 1 \rangle = (x_2 + x_3)\langle 2 \rangle \end{array} \right] \vdash \mathbf{wp} \left[\begin{array}{l} 1: \text{MPSAdd} \\ 2: \text{MPSAdd} \end{array} \right] \left\{ \left[\text{view}_1\langle 1 \rangle = \text{view}_1\langle 2 \rangle \right] \right\}$$

The two specifications look quite different and also suggest quite different proof strategies: the unary judgment suggests a proof by manipulating independence and conditioning; the relational one hints at a proof by relational lifting. Depending on the program, each of these strategies could have their merits. As a first contribution, we show that BLUEBELL can not only specify in both styles (RQ1), but also provide *proofs* in both styles (RQ4).

Having two very different specifications for the same security goal, however begs the question: are they equivalent? After all, as the *prover* of the property one might prefer one proof style over the other, but as a *consumer* of the specification the choice might be dictated by the proof context that needs to use the specification for proving a global goal. To decouple the proof strategy from the uses of the specification, we would need to be able to convert one specification into the other *within* the logic, thus sparing the prover from having to foresee which specification a proof context might need in the future.

Our second key result is that in fact the equivalence between the unary and the relational specification can be proven in BLUEBELL. This is enabled by the powerful joint conditioning rules and the encoding of relational lifting as joint conditioning. This is remarkable as this type of result

has always been justified entirely at the level of the semantic model in other logics (e.g. pRHL, Lilac). This illustrates the fitness of BLUEBELL as a tool for abstract meta-level reasoning (RQ1).

In Section G.5 we provide BLUEBELL proofs for: (1) the unary specification; (2) the relational specification (independently of the unary proof); (3) the equivalence of the two specifications. Although the third item would spare us from proving one of the first two, we provide direct proofs in the two styles to provide a point of comparison between them.

5.5 Von Neumann Extractor

A randomness extractor is a mechanism that transforms a stream of “low-quality” randomness sources into a stream of “high-quality” randomness sources. The von Neumann extractor [von Neumann 1951] is perhaps the earliest instance of such mechanism, and it converts a stream of independent coins with the same bias p into a stream of independent fair coins. Verifying the correctness of the extractor requires careful reasoning under conditioning, and showcases the use of Section 5.1 in a unary setting (RQ2 and RQ4).

We can model the extractor, up to $N \in \mathbb{N}$ iterations, in our language³ as shown in Fig. 5. The program repeatedly flips two biased coins, and outputs the outcome of the first coin if the outcomes where different, otherwise it retries. As an example, we prove in BLUEBELL that the bits produced in out are independent fair coin flips. Formally, for ℓ produced bits, we want the following to hold:

$$\text{Out}_\ell \triangleq \text{out}[0]\langle 1 \rangle \sim \text{Ber}_{1/2} * \dots * \text{out}[\ell - 1]\langle 1 \rangle \sim \text{Ber}_{1/2}.$$

To know how many bits were produced, however, we need to condition on len obtaining the specification (recall $P \Vdash Q \triangleq P \wedge \text{own}_{\mathbb{X}} \vdash Q \wedge \text{own}_{\mathbb{X}}$):

$$\vdash \text{wp} [1: \text{vn}(N)] \{ \exists \mu. C_\mu \ell. (\lceil \text{len}\langle 1 \rangle = \ell \leq N \rceil * \text{Out}_\ell) \}$$

The postcondition straightforwardly generalizes to a loop invariant

$$P(i) = \exists \mu. C_\mu \ell. (\lceil \text{len}\langle 1 \rangle = \ell \leq i \rceil * \text{Out}_\ell)_{\text{len} := 0}$$

The main challenge in the example is handling the if-then statement. Intuitively we want to argue that if $\text{coin}_1 \neq \text{coin}_2$, the two coins would have either values (0, 1) or (1, 0), and both of these outcomes have probability $p(1 - p)$; therefore, conditionally on the ‘if’ guard being true, coin_1 is a fair coin.

Two features of BLUEBELL are crucial to implement the above intuition. The first is the ability of manipulating conditioning given by the joint conditioning rules. At the entry point of the if-then statement in Fig. 5 we obtain $P(i) * \text{coin}_1\langle 1 \rangle \sim \text{Ber}_p * \text{coin}_2\langle 1 \rangle \sim \text{Ber}_p$. Using BLUEBELL’s rules we can easily derive $P(i) * (\text{coin}_1 \neq \text{coin}_2, \text{coin}_1)\langle 1 \rangle \sim \mu_0$ for some μ_0 . The main insight of the algorithm then can be expressed as the fact that $\mu_0 = \beta \prec \kappa$ for some $\beta: \mathbb{D}(\{0, 1\})$ which is the distribution of $\text{coin}_1 \neq \text{coin}_2$, and some κ describing the distribution of the first coin in the two cases, which we know is such that $\kappa(1) = \text{Ber}_{1/2}$. Then, thanks to C-UNIT-R and C-FUSE, we obtain:

$$\begin{aligned} & (\text{coin}_1 \neq \text{coin}_2, \text{coin}_1)\langle 1 \rangle \sim (\beta \prec \kappa) \\ & \vdash C_\beta b. (\lceil (\text{coin}_1 \neq \text{coin}_2)\langle 1 \rangle = b \rceil * \lceil b = 1 \rceil \Rightarrow \text{coin}_1\langle 1 \rangle \sim \text{Ber}_{1/2}) \end{aligned}$$

Then, if we could reason about the ‘then’ branch under conditioning, since the guard $\text{coin}_1 \neq \text{coin}_2$ implies $b = 1$ we would obtain $\text{coin}_1\langle 1 \rangle \sim \text{Ber}_{1/2}$, which is the key to the proof. The ability of reasoning under conditioning is the second feature of BLUEBELL which unlocks the proof. In this

³While technically our language does not support arrays, they can be easily encoded as a collection of N variables.

def vn(N):

repeat N :
 $\text{coin}_1 \approx \text{Ber}(p)$
 $\text{coin}_2 \approx \text{Ber}(p)$
if $\text{coin}_1 \neq \text{coin}_2$ **then**:
 $\text{out}[\text{len}] := \text{coin}_1$
 $\text{len} := \text{len} + 1$

Fig. 5. Von Neumann extractor.

case, the step is driven by Section 5.1, which allows us to prove the if-then statement by case analysis on b .

5.6 Monte Carlo Algorithms

By elaborating on the Monte Carlo example of Section 1, we want to show the fitness of BLUEBELL as a program logic (RQ4) and its specific approach for dealing with the structure of a program. Recall the example in Figure 1 and the goal outlined in Section 1 of comparing the accuracy of the two Monte Carlo algorithms BETW_SEQ and BETW. This goal can be encoded as

$$\left(\begin{array}{l} \llbracket 1\langle 1 \rangle = r\langle 1 \rangle = 0 \rrbracket * \\ \llbracket 1\langle 2 \rangle = r\langle 2 \rangle = 0 \rrbracket \end{array} \right) @p \vdash \mathbf{wp} \left[\begin{array}{l} 1: \text{BETW_SEQ}(x, S) \\ 2: \text{BETW}(x, S) \end{array} \right] \left\{ \llbracket d\langle 1 \rangle \leq d\langle 2 \rangle \rrbracket \right\}$$

(where p contains full permissions for all the variables) which, through the relational lifting, states that it is more likely to get a positive answer from BETW than from BETW_SEQ. The challenge is implementing the intuitive relational argument sketched in Section 1, in the presence of very different looping structures. More precisely, we want to compare the sequential composition of two loops $l_1 = (\mathbf{repeat} \ N \ t_A; \mathbf{repeat} \ N \ t_B)$ with a single loop $l_2 = \mathbf{repeat} \ (2N) \ t$ considering the N iterations of t_A in lockstep with the first N iterations of l_2 , and the N iterations of t_B with the remaining N iterations of l_2 . It is not possible to perform such proof purely in pRHL, which can only handle loops that are perfectly aligned, and tools based on pRHL overcome this limitation by offering a number of code transformations, proved correct externally to the logic, with which one can rewrite the loops so that they syntactically align. In this case such a transformation could look like $\mathbf{repeat} \ (M + N) \ t \equiv \mathbf{repeat} \ M \ t; \mathbf{repeat} \ N \ t$, using which one can rewrite l_2 so it aligns with the two shorter loops. What BLUEBELL can achieve is to avoid the use of such ad-hoc syntactic transformations, and produce a proof structured in two steps: first, one can prove, *within the logic*, that it is sound to align the loops as described; and then proceed with the proof of the aligned loops.

The key idea is that the desired alignment of loops can be expressed as a (derived) rule, encoding the net effect of the syntactic loop splitting, without having to manipulate the syntax:

WP-LOOP-SPLIT

$$\frac{\begin{array}{c} P_1(N_1) \vdash P_2(0) \\ \forall i < N_1. P_1(i) \vdash \mathbf{wp} \ [1: t_1, 2: t] \ \{P_1(i+1)\} \\ \forall j < N_2. P_2(j) \vdash \mathbf{wp} \ [1: t_2, 2: t] \ \{P_2(j+1)\} \end{array}}{P_1(0) \vdash \mathbf{wp} \ [1: (\mathbf{repeat} \ N_1 \ t_1; \mathbf{repeat} \ N_2 \ t_2), 2: \mathbf{repeat} \ (N_1 + N_2) \ t] \ \{P_2(N_2)\}}$$

The rule considers two programs: a sequence of two loops, and a single loop with the same cumulative number of iterations. It asks the user to produce two relational loop invariants P_1 and P_2 which are used to relate N_1 iterations of t_1 and t together, and N_2 iterations of t_2 and t together.

Crucially, such rule is *derivable* from the primitive rules of looping of BLUEBELL:

WP-LOOP

$$\frac{\forall i < n. P(i) \vdash \mathbf{wp} \ [j: t] \ \{P(i+1)\}}{P(0) \vdash \mathbf{wp} \ [j: \mathbf{repeat} \ n \ t] \ \{P(n)\}} \quad n \in \mathbb{N}$$

WP-LOOP-UNF

$$\frac{\mathbf{wp} \ [i: \mathbf{repeat} \ n \ t] \ \{\mathbf{wp} \ [i: t] \ \{Q\}\}}{\vdash \mathbf{wp} \ [i: \mathbf{repeat} \ (n+1) \ t] \ \{Q\}}$$

Section 5.6 is a standard unary invariant-based rule; WP-LOOP-UNF simply reflects the semantics of a loop in terms of its unfoldings. Using these we can prove WP-LOOP-SPLIT avoiding semantic reasoning all together, and fully generically on the loop bodies, allowing it to be reused in any situation fitting the pattern.

In our example, we can prove our goal by instantiating it with the loop invariants:

$$P_1(i) \triangleq \llbracket r\langle 1 \rangle \leq r\langle 2 \rangle \wedge 1\langle 1 \rangle = 0 \leq 1\langle 2 \rangle \rrbracket \quad P_2(j) \triangleq \llbracket r\langle 1 \rangle \leq r\langle 2 \rangle \wedge 1\langle 1 \rangle \leq 1\langle 2 \rangle \rrbracket$$

```

def BETW_MIX(x, S) :
  repeat N :
    p ≈ μS; l := 1 || p ≤ x
    q ≈ μS; r := r || q ≥ x
  d := r && l

```

Fig. 6. A variant of the BETW program.

```

def prog1:    def prog2:
  x ≈ d0      x ≈ d0
  y ≈ d1(x)    z ≈ d2(x)
  z ≈ d2(x)    y ≈ d1(x)

```

Fig. 7. Conditional Swapping

This handling of structural differences as derived proof patterns is more powerful than syntactic transformations: it can, for example, handle transformations that are sound only under some assumptions about state. To show an instance of this, we consider a variant of the previous example: BETW_MIX (in Fig. 6) is another variant of BETW_SEQ which still makes $2N$ samples but interleaves sampling for the minimum and for the maximum. We want to prove that this is equivalent to BETW_SEQ. Letting \mathbf{p} contain full permissions for the relevant variables, the goal is

$$P_0 @ \mathbf{p} \vdash \mathbf{wp} [\mathbf{1}: \text{BETW_SEQ}(x, S), \mathbf{2}: \text{BETW_MIX}(x, S)] \{ \lfloor d\langle 1 \rangle = d\langle 2 \rangle \rfloor \}$$

with $P_0 = \lfloor l\langle 1 \rangle = r\langle 1 \rangle = 0 \rfloor * \lfloor l\langle 2 \rangle = r\langle 2 \rangle = 0 \rfloor$.

Call t_M^1 and t_M^2 the first and second half of the body of the loop of BETW_MIX, respectively. The strategy is to consider together one execution of t_A (the body of the loop of AboveMin), and t_M^1 ; and one of t_B (of BelowMax), and t_M^2 . The strategy relies on the observation that every iteration of the three loops is *independent* from the others. To formalize the proof idea we thus first prove a derived proof pattern encoding the desired alignment, which we can state for generic t_1, t_2, t'_1, t'_2 :

WP-LOOP-MIX

$$\frac{\forall i < N. P_1(i) \vdash \mathbf{wp} [\mathbf{1}: t_1, \mathbf{2}: t'_1] \{P_1(i+1)\} \quad \forall i < N. P_2(i) \vdash \mathbf{wp} [\mathbf{1}: t_2, \mathbf{2}: t'_2] \{P_2(i+1)\}}{P_1(0) * P_2(0) \vdash \mathbf{wp} [\mathbf{1}: (\text{repeat } N \ t_1; \text{repeat } N \ t_2), \mathbf{2}: \text{repeat } N \ (t'_1; t'_2)] \{P_1(N) * P_2(N)\}}$$

The rule matches on two programs: a sequence of two loops, and a single loop with a body split into two parts. The premises require a proof that t_1 together with t'_1 (the first half of the body of the second loop) preserve the invariant P_1 ; and that the same is true for t_2 and t'_2 with respect to an invariant P_2 . The precondition $P_1(0) * P_2(0)$ in the conclusion ensures that the two loop invariants are independent. The rule WP-LOOP-MIX can be again entirely derived from BLUEBELL's primitive rules. We can then apply it to our example using as invariants $P_1 \triangleq \lfloor l\langle 1 \rangle = l\langle 2 \rangle \rfloor$ and $P_2 \triangleq \lfloor r\langle 1 \rangle = r\langle 2 \rangle \rfloor$. Then, RL-MERGE closes the proof.

6 Related Work

Research on deductive verification of probabilistic programs has developed a wide range of techniques that employ *unary* and *relational* styles of reasoning. BLUEBELL advances the state of the art in both styles, by coherently unifying the strengths of both. We limit our comparison here to deductive techniques only, and focus most of our attention on explaining how BLUEBELL offers new reasoning tools compared to these.

Unary-style Reasoning. Early work in this line focuses more on analyzing marginal distributions and probabilities, and features like harnessing the power of probabilistic independence and conditioning have been more recently added to make more expressive program logics [Bao et al. 2022; Barthe et al. 2018, 2019; Li et al. 2023a; Ramshaw 1979; Rand and Zdancewic 2015].

Much work in this line has been inspired by *Separation Logic* (SL), a powerful tool for reasoning about pointer-manipulating programs, known for its support of *local reasoning* of separated program

components [Reynolds 2000]. PSL [Barthe et al. 2019] was the first logic to present a SL model for reasoning about the probabilistic independence of program variables, which facilitates modular reasoning about independent components within a probabilistic program. In [Bao et al. 2021] and [Bao et al. 2022] SL variants are used for reasoning about conditional independence and negative dependence, respectively; both are used in algorithm analysis as relaxations of independence.

Lilac. Lilac [Li et al. 2023a] is the most recent addition to this group and introduces a new foundation of probabilistic separation logic based on measure theory. It enables reasoning about independence and conditional independence uniformly in one logic and supports continuous distributions. BLUEBELL also uses a measure-theory based model, similar to Lilac, although limited to discrete distributions. While BLUEBELL uses Lilac’s independent product as a model of separating conjunction, it differs from Lilac in three aspects: (1) the treatment of ownership, (2) support for mutable state, and (3) the model of conditioning.

Ownership as almost-measurability is required to support inferences like $\text{own}(x) * [x = y] \vdash \text{own}(y)$, which were implicitly used in the first version of Lilac, but were not valid in its model. Li et al. [2023b] fixes the issue by changing the meaning of $[x = y]$, while our fix acts on the meaning of ownership (and we see $[E]$ assertions as an instance of regular ownership).

Lilac works with immutable state [Staton 2020], which simplifies reasoning in certain contexts (e.g., the frame rule and the if rule). BLUEBELL’s model supports mutable state through a creative use of permissions, obtaining a clean frame rule, at the cost of some predictable bookkeeping.

The more significant difference with Lilac is however in the definition of the conditioning modality. Lilac’s modality $C_{v \leftarrow E} P(v)$ is indexed by a random variable E , and roughly corresponds to the BLUEBELL assertion $\exists \mu. C_\mu v. ([E = v] * P(v))$. The difference is not merely syntactic, and requires changing the model of the modality. For example, Lilac’s modality satisfies $C_{v \leftarrow E} P_1(v) \wedge C_{v \leftarrow E} P_2(v) \vdash C_{v \leftarrow E} (P_1(v) \wedge P_2(v))$, but the analogous rule $C_\mu v. K_1(v) \wedge C_\mu v. K_2(v) \vdash C_\mu v. (K_1(v) \wedge K_2(v))$ (corresponding to **C-AND** without the side condition) is unsound in BLUEBELL: The meaning of the modalities in the premise ensures the *existence* of two kernels κ_1 and κ_2 supporting K_1 and K_2 respectively, but the conclusion requires the existence of a *single* kernel supporting both K_1 and K_2 . Lilac’s rule holds because when one conditions on a random variable, the corresponding kernels are unique. We did not find losing this rule limiting. On the other hand, Lilac’s conditioning has two key disadvantages: (i) it does not record the distribution of E , losing this information when conditioning, (ii) it does not generalize to the relational setting. Even considering only the unary setting, having access to the distribution μ in fact unlocks a number of new rules (e.g. **C-UNIT-R** and **C-FUSE**) that are key to the increased expressivity of BLUEBELL. In particular, the rules of BLUEBELL provide a wider arsenal of tools that can convert a conditional assertion back into an unconditional one. This is especially important when conditioning is used as a reasoning tool, regardless of whether the end goal is a conditional statement.

Relational Reasoning. Barthe et al. [2009] extend relational Hoare logic [Benton 2004] to reason about probabilistic programs in a logic called pRHL (probabilistic Relational Hoare Logic). In pRHL, assertions on pairs of deterministic program states are lifted to assertions on pairs of distributions, and on the surface, the logic simply manipulates the deterministic assertions. A number of variants of pRHL were successfully applied to proving various cryptographic protocols and differential privacy algorithms [Barthe et al. 2015, 2009; Hsu 2017; Wang et al. 2019; Zhang and Kifer 2017]. When a natural relational proof for an argument exists, these logics are simple and elegant to use. However, they fundamentally trade expressiveness for ease of use. A persisting problem with them has been that they rely on a strict structural alignment between the order of samples in the two programs. Recall our discussion in Section 2.4 for an example of this that BLUEBELL can

handle. [Gregersen et al. \[2024\]](#) recently proposed Clutch, a logic to prove contextual refinement in a probabilistic higher-order language, where “out of order” couplings between samplings are achieved by using ghost code that pre-samples some assignments, a technique inspired by *prophecy variables* [\[Jung et al. 2019\]](#). In Section 2 we showed how BLUEBELL can resolve the issue without ghost code (in the context of first-order imperative programs) by using framing and probabilistic independence creatively. In contrast to BLUEBELL, Clutch can only express relational properties; it also uses separation but with its classical interpretation as disjointness of deterministic state.

Polaris [\[Tassarotti and Harper 2019\]](#), is an early instance of a probabilistic relational (concurrent) separation logic. However, separation in Polaris is again classic disjointness of state.

Our n -ary WP is inspired by LHC [\[D’Oswaldo et al. 2022\]](#), which shows how arity-changing rules (like **WP-NEST**) can accommodate modular and flexible relational proofs of deterministic programs.

Other Techniques. Expectation-based approaches, which reason about expected quantities of probabilistic programs via a weakest-pre-expectation operator that propagates information about expected values backwards through the program, have been classically used to verify randomized algorithms [\[Aguirre et al. 2021; Kaminski 2019; Kaminski et al. 2016; Kozen 1983; Moosbrugger et al. 2022; Morgan et al. 1996\]](#). These logics offer ergonomic dedicated principles for expectations, but do not aim at unifying principles for analyzing more general classes of properties or proof techniques, like we attempt here. Ellora [\[Barthe et al. 2018\]](#) proposes an assertion-based logic (without separation nor conditioning) to overcome the limitation of working only with expectations.

7 Conclusions and Future Work

BLUEBELL’s journey started as a quest to integrate unary and relational probabilistic reasoning and ended up uncovering joint conditioning as a key foundational tool. Remarkably, to achieve our goal we had to deviate from Lilac’s previous proposal in both the definition of conditioning, to enable the encoding of relational lifting, and of ownership (with almost measurability), to resolve an issue with almost sure assertions (recently corrected [\[Li et al. 2023b\]](#) in a different way). In addition, our model supports mutable state without sacrificing expressiveness. One limitation of our current model is lack of support for continuous distributions. Lilac’s model and recent advances in it [\[Li et al. 2024\]](#) could suggest a pathway for a continuous extension of BLUEBELL, but it is unclear if all our rules would be still valid; for example Section 2.3’s soundness hinges on properties of discrete distributions that we could not extend to the general case in an obvious way. BLUEBELL’s encoding of relational lifting and the novel proof principles it uncovered for it are a demonstration of the potential of joint conditioning as a basis for deriving high-level logics on top of an ergonomic core logic. Obvious candidates for such scheme are approximate couplings [\[Barthe et al. 2012\]](#) (which have been used for e.g. differential privacy), and expectation-based calculi (à la Ellora).

Acknowledgments

We would like to thank Justin Hsu for connecting the authors and the many discussions. We also thank Derek Dreyer and Deepak Garg for the discussions and support. We are grateful to the POPL’25 reviewers for their constructive feedback. Jialu Bao was supported by the NSF Award No. 2153916. Emanuele D’Oswaldo was supported by a European Research Council (ERC) Consolidator Grant for the project “PERSIST” under the European Union’s Horizon 2020 research and innovation programme (grant No. 101003349). Azadeh Farzan was supported by the National Science and Engineering Research Council of Canada Discovery Grant.

References

- Alejandro Aguirre, Gilles Barthe, Marco Gaboardi, Deepak Garg, and Pierre-Yves Strub. 2017. A relational logic for higher-order programs. *Proceedings of the ACM on Programming Languages* 1, ACM SIGPLAN International Conference on Functional Programming (ICFP), Oxford, England (2017), 1–29. <https://doi.org/10.1145/3110265>
- Alejandro Aguirre, Gilles Barthe, Justin Hsu, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. 2021. A pre-expectation calculus for probabilistic sensitivity. *Proceedings of the ACM on Programming Languages* 5, ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), Lisbon, Portugal (2021), 1–28. <https://doi.org/10.1145/3434333>
- Jialu Bao, Simon Docherty, Justin Hsu, and Alexandra Silva. 2021. A bunched logic for conditional independence. In *IEEE Symposium on Logic in Computer Science (LICS), Rome, Italy*. IEEE, 1–14. <https://doi.org/10.1109/LICS52264.2021.9470712>
- Jialu Bao, Marco Gaboardi, Justin Hsu, and Joseph Tassarotti. 2022. A separation logic for negative dependence. *Proceedings of the ACM on Programming Languages* 6, ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), Philadelphia, Pennsylvania (2022), 1–29. <https://doi.org/10.1145/3498719>
- Gilles Barthe, Thomas Espitau, Marco Gaboardi, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. 2018. An assertion-based program logic for probabilistic programs. In *Programming Languages and Systems*. Springer International Publishing, Cham, 117–144.
- Gilles Barthe, Thomas Espitau, Benjamin Grégoire, Justin Hsu, Léo Stefanescu, and Pierre-Yves Strub. 2015. Relational reasoning via probabilistic coupling. In *International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR), Suva, Fiji*. Springer, 387–401. https://doi.org/10.1007/978-3-662-48899-7_27
- Gilles Barthe, Benjamin Grégoire, and Santiago Zanella Béguelin. 2009. Formal certification of code-based cryptographic proofs. In *ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), Savannah, Georgia*. 90–101. <https://doi.org/10.1145/1480881.1480894>
- Gilles Barthe, Justin Hsu, and Kevin Liao. 2019. A probabilistic separation logic. *Proceedings of the ACM on Programming Languages* 4, ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), Lisbon, Portugal (2019), 1–30. <https://doi.org/10.1145/3371123>
- Gilles Barthe, Boris Köpf, Federico Olmedo, and Santiago Zanella Béguelin. 2012. Probabilistic relational reasoning for differential privacy. In *ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), Philadelphia, Pennsylvania*. ACM, 97–110. <https://doi.org/10.1145/2103656.2103670>
- Nick Benton. 2004. Simple relational correctness proofs for static analyses and program transformations. In *ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), Venice, Italy*, Vol. 39. ACM New York, NY, USA, 14–25. <https://doi.org/10.1145/964001.964003>
- Richard Bornat, Cristiano Calcagno, and Hongseok Yang. 2005. Variables as resource in separation logic. In *Conference on the Mathematical Foundations of Programming Semantics (MFPS), Birmingham, England (Electronic Notes in Theoretical Computer Science, Vol. 155)*. Elsevier, 247–276. <https://doi.org/10.1016/J.ENTCS.2005.11.059>
- Emanuele D’Osualdo, Azadeh Farzan, and Derek Dreyer. 2022. Proving hypersafety compositionally. *Proceedings of the ACM on Programming Languages* 6, ACM SIGPLAN Conference on Object Oriented Programming: Systems, Languages, and Applications (OOPSLA), Auckland, New Zealand (2022), 289–314. <https://doi.org/10.1145/3563298>
- Michele Giry. 1982. A categorical approach to probability theory. In *Categorical Aspects of Topology and Analysis: Proceedings of an International Conference Held at Carleton University, Ottawa, August 11–15, 1981*. Springer, 68–85. <https://doi.org/10.1007/S11225-010-9232-Z>
- Simon Oddershede Gregersen, Alejandro Aguirre, Philipp G. Haselwarter, Joseph Tassarotti, and Lars Birkedal. 2024. Asynchronous probabilistic couplings in higher-order separation logic. *Proceedings of the ACM on Programming Languages* 8, ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), London, England (2024), 753–784. <https://doi.org/10.1145/3632868>
- Justin Hsu. 2017. *Probabilistic couplings for probabilistic reasoning*. Ph.D. Dissertation.
- Ralf Jung, Rodolphe Lepigre, Gaurav Parthasarathy, Marianna Rapoport, Amin Timany, Derek Dreyer, and Bart Jacobs. 2019. The future is ours: prophecy variables in separation logic. *Proceedings of the ACM on Programming Languages* 4, ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), Lisbon, Portugal (2019), 1–32. <https://doi.org/10.1145/3371113>
- Benjamin Lucien Kaminski. 2019. *Advanced weakest precondition calculi for probabilistic programs*. Ph.D. Dissertation. RWTH Aachen University.
- Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Federico Olmedo. 2016. Weakest precondition reasoning for expected run-times of probabilistic programs. In *European Symposium on Programming (ESOP), Eindhoven, The Netherlands*. Springer, 364–389. https://doi.org/10.1007/978-3-662-49498-1_15
- Dexter Kozen. 1983. A Probabilistic PDL. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 25-27 April, 1983, Boston, Massachusetts, USA*. ACM, 291–297. <https://doi.org/10.1145/800061.808758>

- Robbert Krebbers, Jacques-Henri Jourdan, Ralf Jung, Joseph Tassarotti, Jan-Oliver Kaiser, Amin Timany, Arthur Charguéraud, and Derek Dreyer. 2018. MoSeL: a general, extensible modal framework for interactive proofs in separation logic. *Proceedings of the ACM on Programming Languages* 2 (2018), 77:1–77:30. <https://doi.org/10.1145/3236772>
- John M Li, Amal Ahmed, and Steven Holtzen. 2023a. Lilac: a modal separation logic for conditional probability. *Proceedings of the ACM on Programming Languages* 7 (2023), 148–171. <https://doi.org/10.1145/3591226>
- John M. Li, Amal Ahmed, and Steven Holtzen. 2023b. Lilac: A Modal Separation Logic for Conditional Probability. [arXiv:2304.01339v2](https://arxiv.org/abs/2304.01339v2)
- John M. Li, Jon Aytac, Philip Johnson-Freyd, Amal Ahmed, and Steven Holtzen. 2024. A nominal approach to probabilistic separation logic. In *IEEE Symposium on Logic in Computer Science (LICS), Tallinn, Estonia*. ACM, 55:1–55:14. <https://doi.org/10.1145/3661814.3662135>
- Marcel Moosbrugger, Miroslav Stanković, Ezio Bartocci, and Laura Kovács. 2022. This is the moment for probabilistic loops. *Proceedings of the ACM on Programming Languages* 6, ACM SIGPLAN Conference on Object Oriented Programming: Systems, Languages, and Applications (OOPSLA), Auckland, New Zealand (2022), 1497–1525. <https://doi.org/10.1145/3563341>
- Carroll Morgan, Annabelle McIver, and Karen Seidel. 1996. Probabilistic predicate transformers. *ACM Transactions on Programming Languages and Systems* (1996). <https://doi.org/10.1145/229542.229547>
- Lyle Harold Ramshaw. 1979. *Formalizing the analysis of algorithms*. Vol. 75. Xerox Palo Alto Research Center.
- Robert Rand and Steve Zdancewic. 2015. VPHL: A verified partial-correctness logic for probabilistic programs. *Electronic Notes in Theoretical Computer Science* 319 (2015), 351–367. <https://doi.org/10.1016/J.ENTCS.2015.12.021>
- John Reynolds. 2000. Intuitionistic reasoning about shared mutable data structure. *Millennial perspectives in computer science* 2, 1 (2000), 303–321.
- Jeffrey S. Rosenthal. 2006. *A first look at rigorous probability theory*. World Scientific Publishing Company.
- Sam Staton. 2020. Probabilistic programs as measures. *Foundations of Probabilistic Programming* (2020), 43. <https://doi.org/10.1017/9781108770750>
- Joseph Tassarotti and Robert Harper. 2019. A separation logic for concurrent randomized programs. *Proceedings of the ACM on Programming Languages* 3, ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), Lisbon, Portugal (2019), 1–30. <https://doi.org/10.1145/3290377>
- Jeffrey S Vitter. 1985. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)* 11, 1 (1985), 37–57. <https://doi.org/10.1145/3147.3165>
- John von Neumann. 1951. Various techniques used in connection with random digits. *Journal of Research of the National Bureau of Standards, Applied Math Series* (1951), 36–38.
- Yuxin Wang, Zeyu Ding, Guanhong Wang, Daniel Kifer, and Danfeng Zhang. 2019. Proving differential privacy with shadow execution. *Proceedings of the ACM on Programming Languages* ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), Lisbon, Portugal (2019), 655–669. <https://doi.org/10.1145/3314221.3314619>
- Danfeng Zhang and Daniel Kifer. 2017. LightDP: Towards automating differential privacy proofs. In *ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), Paris, France*. 888–901. <https://doi.org/10.1145/3009837.3009884>

Appendix

A The Rules of BLUEBELL

In this section we list all the rules of BLUEBELL, including some omitted (but useful) rules in addition to those that appear in the main text. Since our treatment is purely semantic, rules are simply lemmas that hold in the model. Although we do not aim for a full axiomatization, we try to identify the key proof principles that apply to each of our connectives. For brevity, we omit the rules that apply to the basic connectives of separation logic, as they are well-known and have been proven correct for any model that is an RA. For those we refer to [Krebbbers et al. 2018].

In Fig. 8 we summarize the notation we use for assertions over BLUEBELL’s model. Recall that BLUEBELL’s assertions $P \in \text{HA}_I \triangleq \mathcal{M}_I \xrightarrow{u} \text{Prop}$ are the upward-closed predicates over elements of the RA \mathcal{M}_I .

$$\begin{aligned}
 \ulcorner \phi \urcorner &\triangleq \lambda_{-}. \phi \\
 \text{Own}(b) &\triangleq \lambda a. b \preceq a \\
 P \wedge Q &\triangleq \lambda a. P(a) \wedge Q(a) \\
 P * Q &\triangleq \lambda a. \exists b_1, b_2. (b_1 \cdot b_2) \preceq a \wedge P(b_1) \wedge Q(b_2) \\
 \exists x : X. K &\triangleq \lambda a. \exists x : X. K(x)(a) & (K : X \rightarrow \text{HA}_I) \\
 \forall x : X. K &\triangleq \lambda a. \forall x : X. K(x)(a) & (K : X \rightarrow \text{HA}_I) \\
 \text{Own}(\mathcal{F}, \mu) &\triangleq \exists p. \text{Own}(\mathcal{F}, \mu, p) \\
 E\langle i \rangle \sim \mu &\triangleq \exists \mathcal{F}, \mu. \text{Own}(\mathcal{F}, \mu) * \ulcorner E \prec (\mathcal{F}(i), \mu(i)) \wedge \mu = \mu(i) \circ E^{-1} \urcorner \\
 C_\mu K &\triangleq \lambda a. \exists \mathcal{F}, \mu, p, \kappa. (\mathcal{F}, \mu, p) \preceq a \wedge \forall i \in I. \mu(i) = \text{bind}(\mu, \kappa(i)) & (\mu : \mathbb{D}(A), K : A \rightarrow \text{HA}_I) \\
 &\quad \wedge \forall v \in \text{supp}(\mu). K(v)(\mathcal{F}, \kappa(I)(v), p) \\
 \text{wp } t \{Q\} &\triangleq \lambda a. \forall \mu_0. \forall c. (a \cdot c) \preceq \mu_0 \Rightarrow \exists b. ((b \cdot c) \preceq \llbracket t \rrbracket(\mu_0) \wedge Q(b)) \\
 \llbracket E\langle i \rangle \rrbracket &\triangleq (E \in \text{true})\langle i \rangle \sim \delta_{\text{true}} \\
 \text{own}(E\langle i \rangle) &\triangleq \exists \mu. E\langle i \rangle \sim \mu \\
 (x\langle i \rangle : q) &\triangleq \exists \mathcal{P}, p. \text{Own}(\mathcal{P}, p) * \ulcorner p(i)(x) = q \urcorner \\
 P @ p &\triangleq P \wedge \exists \mathcal{P}. \text{Own}(\mathcal{P}, p) \\
 \llbracket R \rrbracket &\triangleq \exists \mu : \mathbb{D}(\mathbb{V}^X). \ulcorner \mu(R) = 1 \urcorner * C_\mu v. \llbracket x\langle i \rangle = v(x\langle i \rangle) \rrbracket_{x\langle i \rangle \in E} & (R \subseteq \mathbb{V}^X, X \subseteq I \times \mathbb{X})
 \end{aligned}$$

Fig. 8. The assertions used in BLUEBELL.

PROPOSITION A.1 (UPWARD-CLOSURE). *All the assertions in Fig. 8 are upward-closed.*

PROOF. Easy by inspection of the definitions. The definitions where upward-closedness is less obvious (e.g. joint conditioning) are made upward-closed by construction by explicit use of the order \preceq in the definition. \square

Although we adopt a “shallow embedding” approach to assertions (and thus we do not provide separate syntax), the rules of BLUEBELL provide an axiomatic treatment of these assertions so that the user should never manipulate raw predicates over the semantic model. We consider the connectives listed above WP (included) in Fig. 8 to be the ones that the user should never need to unfold into their definitions and only manipulate through rules.

We make a distinction between “primitive” and “derived” rules. The primitive rules require proofs that manipulate the semantic model definitions directly; these are the ones we would consider part

— Distribution ownership rules —

AND-TO-STAR

$$\frac{\text{idx}(P) \cap \text{idx}(Q) = \emptyset}{P \wedge Q \vdash P * Q}$$

DIST-INJ

$$E\langle i \rangle \sim \mu \wedge E\langle i \rangle \sim \mu' \vdash \ulcorner \mu = \mu' \urcorner$$

SURE-MERGE

$$\llbracket E_1\langle i \rangle \rrbracket * \llbracket E_2\langle i \rangle \rrbracket \dashv\vdash \llbracket (E_1 \wedge E_2)\langle i \rangle \rrbracket$$

SURE-AND-STAR

$$\frac{\text{pabs}(P, \text{pvar}(E\langle i \rangle))}{\llbracket E\langle i \rangle \rrbracket \wedge P \vdash \llbracket E\langle i \rangle \rrbracket * P}$$

PROD-SPLIT

$$(E_1\langle i \rangle, E_2\langle i \rangle) \sim \mu_1 \otimes \mu_2 \vdash E_1\langle i \rangle \sim \mu_1 * E_2\langle i \rangle \sim \mu_2$$

— Joint conditioning rules —

C-TRUE

$$\vdash C_\mu _ . \text{True}$$

C-FALSE

$$C_\mu v . \text{False} \vdash \text{False}$$

C-CONS

$$\frac{\forall v . K_1(v) \vdash K_2(v)}{C_\mu v . K_1(v) \vdash C_\mu v . K_2(v)}$$

C-FRAME

$$P * C_\mu v . K(v) \vdash C_\mu v . (P * K(v))$$

C-UNIT-L

$$C_{\delta_{v_0}} v . K(v) \dashv\vdash K(v_0)$$

C-UNIT-R

$$E\langle i \rangle \sim \mu \dashv\vdash C_\mu v . \llbracket E\langle i \rangle = v \rrbracket$$

C-ASSOC

$$\frac{\mu_0 = \text{bind}(\mu, \lambda v . (\text{bind}(\kappa(v), \lambda w . \text{return}(v, w))))}{C_\mu v . C_{\kappa(v)} w . K(v, w) \vdash C_{\mu_0}(v, w) . K(v, w)}$$

C-UNASSOC

$$C_{\text{bind}(\mu, \kappa)} w . K(w) \vdash C_\mu v . C_{\kappa(v)} w . K(w)$$

C-AND

$$\frac{\text{idx}(K_1) \cap \text{idx}(K_2) = \emptyset}{C_\mu v . K_1(v) \wedge C_\mu v . K_2(v) \vdash C_\mu v . (K_1(v) \wedge K_2(v))}$$

C-SKOLEM

$$\frac{\mu : \mathbb{D}(\Sigma_A)}{C_\mu v . \exists x : X . Q(v, x) \vdash \exists f : A \rightarrow X . C_\mu v . Q(v, f(v))}$$

C-TRANSF

$$\frac{\begin{array}{l} f : \text{supp}(\mu') \rightarrow \text{supp}(\mu) \text{ bijective} \\ \forall b \in \text{supp}(\mu') . \mu'(b) = \mu(f(b)) \end{array}}{C_\mu a . K(a) \vdash C_{\mu'} b . K(f(b))}$$

SURE-STR-CONVEX

$$C_\mu v . (K(v) * \llbracket E\langle i \rangle \rrbracket) \vdash \llbracket E\langle i \rangle \rrbracket * C_\mu v . K(v)$$

C-FOR-ALL

$$C_\mu v . \forall x : X . Q(v, x) \vdash \forall x : X . C_\mu v . Q(v, x)$$

C-PURE

$$\ulcorner \mu(X) = 1^\top * C_\mu v . K(v) \dashv\vdash C_\mu v . (\ulcorner v \in X^\top * K(v) \urcorner)$$

Fig. 9. The primitive rules of BLUEBELL.

of a proper axiomatization. The derived rules can be proved sound by staying at the level of the logic, i.e. by using the primitive rules of BLUEBELL.

Figure 9 lists the primitive rules for distribution ownership assertions and for the joint conditioning modality. Figure 10 lists the primitive rules for the weakest precondition modality. In Fig. 11 we list some useful derived rules, and in Fig. 12 we show some derived rules for WP.

We provide proofs for each rule in the form of lemmas in Section F. The name labelling each rule is a link to the proof of soundness of the rule.

— Structural WP rules —

WP-CONS

$$\frac{Q \vdash Q'}{\mathbf{wp} \, t \{Q\} \vdash \mathbf{wp} \, t \{Q'\}}$$

WP-FRAME

$$P * \mathbf{wp} \, t \{Q\} \vdash \mathbf{wp} \, t \{P * Q\}$$

WP-NEST

$$\mathbf{wp} \, t_1 \{\mathbf{wp} \, t_2 \{Q\}\} \dashv\vdash \mathbf{wp} \, (t_1 \cdot t_2) \{Q\}$$

WP-CONJ

$$\frac{\text{idx}(Q_1) \cap |t_2| \subseteq |t_1| \quad \text{idx}(Q_2) \cap |t_1| \subseteq |t_2|}{\mathbf{wp} \, t_1 \{Q_1\} \wedge \mathbf{wp} \, t_2 \{Q_2\} \vdash \mathbf{wp} \, (t_1 + t_2) \{Q_1 \wedge Q_2\}}$$

C-WP-SWAP

$$C_\mu v. \mathbf{wp} \, t \{Q(v)\} \wedge \text{own}_{\mathbb{X}} \vdash \mathbf{wp} \, t \{C_\mu v. Q(v)\}$$

— Program WP rules —

WP-SKIP

$$P \vdash \mathbf{wp} \, [i: \mathbf{skip}] \{P\}$$

WP-SEQ

$$\mathbf{wp} \, [i: t] \{\mathbf{wp} \, [i: t'] \{Q\}\} \vdash \mathbf{wp} \, [i: (t; t')] \{Q\}$$

WP-ASSIGN

$$\frac{x \notin \text{pvar}(e) \quad \forall y \in \text{pvar}(e). p(y(i)) > 0 \quad p(x(i)) = 1}{(p) \vdash \mathbf{wp} \, [i: x := e] \{[x(i) = e(i)] @ p\}}$$

WP-SAMP

$$(x(i):1) \vdash \mathbf{wp} \, [i: x \approx d(\vec{v})] \{x(i) \sim d(\vec{v})\}$$

WP-IF-PRIM

$$\frac{\text{if } v \text{ then } \mathbf{wp} \, [i: t_1] \{Q(1)\} \text{ else } \mathbf{wp} \, [i: t_2] \{Q(0)\}}{\vdash \mathbf{wp} \, [i: \text{if } v \text{ then } t_1 \text{ else } t_2] \{Q(v)\}}$$

WP-BIND

$$[e(i) = v] * \mathbf{wp} \, [i: \mathcal{E}[v]] \{Q\} \vdash \mathbf{wp} \, [i: \mathcal{E}[e]] \{Q\}$$

WP-LOOP-UNF

$$\frac{\mathbf{wp} \, [i: \mathbf{repeat} \, n \, t] \{\mathbf{wp} \, [i: t] \{Q\}\}}{\vdash \mathbf{wp} \, [i: \mathbf{repeat} \, (n+1) \, t] \{Q\}}$$

WP-LOOP

$$\frac{\forall i < n. P(i) \vdash \mathbf{wp} \, [j: t] \{P(i+1)\}}{P(0) \vdash \mathbf{wp} \, [j: \mathbf{repeat} \, n \, t] \{P(n)\}} \quad n \in \mathbb{N}$$

Fig. 10. The primitive WP rules of BLUEBELL.

B Auxiliary Definitions

Definition B.1 (Bind and return). Let A be a countable set and \mathcal{F} a σ -algebra. We define the following functions:

$$\mathbf{return}: A \rightarrow \mathbb{D}(\Sigma_A)$$

$$\mathbf{return}(a) \triangleq \delta_a$$

$$\mathbf{bind}: \mathbb{D}(\Sigma_A) \rightarrow (A \rightarrow \mathbb{D}(\mathcal{F})) \rightarrow \mathbb{D}(\mathcal{F})$$

$$\mathbf{bind}(\mu, \kappa) \triangleq \lambda X \in \mathcal{F}. \sum_{a \in A} \mu(a) \cdot \kappa(a)(X)$$

We will use throughout Haskell-style notation for monadic expressions, for instance:

$$(x \leftarrow \mu; y \leftarrow f(x); \mathbf{return}(x + y)) \equiv \mathbf{bind}(\mu, \lambda x. \mathbf{bind}(f(x), \lambda y. \mathbf{return}(x + y)))$$

The \mathbf{bind} and \mathbf{return} operators form a well-known monad with \mathbb{D} , and thus satisfy the monadic laws:

$$\mathbf{bind}(\mu, \lambda x. \mathbf{return}(x)) = \mu \quad (\text{UNIT-R})$$

$$\mathbf{bind}(\mathbf{return}(v), \kappa) = \kappa(v) \quad (\text{UNIT-L})$$

$$\mathbf{bind}(\mathbf{bind}(\mu, \kappa_1), \kappa_2) = \mathbf{bind}(\mu, \lambda x. \mathbf{bind}(\kappa_1(x), \kappa_2)) \quad (\text{ASSOC})$$

By Lemma C.1, for any sigma algebra \mathcal{F}' on countable underlying set, there exists a partition S of the underlying space that generated it, so we can transform any such \mathcal{F}' to a full sigma algebra

— Ownership and distributions —

SURE-DIRAC

$$E\langle i \rangle \sim \delta_v \dashv \vdash [E\langle i \rangle = v]$$

SURE-EQ-INJ

$$[E\langle i \rangle = v] * [E\langle i \rangle = v'] \vdash \ulcorner v = v' \urcorner$$

SURE-SUB

$$E_1\langle i \rangle \sim \mu * [E_2 = f(E_1)]\langle i \rangle \vdash E_2\langle i \rangle \sim \mu \circ f^{-1}$$

DIST-FUN

$$E\langle i \rangle \sim \mu \vdash (f \circ E)\langle i \rangle \sim \mu \circ f^{-1}$$

DIRAC-DUP

$$E\langle i \rangle \sim \delta_v \vdash E\langle i \rangle \sim \delta_v * E\langle i \rangle \sim \delta_v$$

DIST-SUPP

$$E\langle i \rangle \sim \mu \vdash E\langle i \rangle \sim \mu * [E\langle i \rangle \in \text{supp}(\mu)]$$

PROD-UNSPLIT

$$E_1\langle i \rangle \sim \mu_1 * E_2\langle i \rangle \sim \mu_2 \vdash (E_1\langle i \rangle, E_2\langle i \rangle) \sim \mu_1 \otimes \mu_2$$

— Joint conditioning —

C-FUSE

$$C_\mu v. C_{\kappa(v)} w. K(v, w) \dashv \vdash C_{\mu \prec \kappa}(v, w). K(v, w)$$

C-SWAP

$$C_{\mu_1} v_1. C_{\mu_2} v_2. K(v_1, v_2) \vdash C_{\mu_2} v_2. C_{\mu_1} v_1. K(v_1, v_2)$$

SURE-CONVEX

$$C_\mu v. [E\langle i \rangle] \vdash [E\langle i \rangle]$$

DIST-CONVEX

$$C_\mu v. E\langle i \rangle \sim \mu' \vdash E\langle i \rangle \sim \mu'$$

C-SURE-PROJ

$$C_\mu(v, w). [E(v)\langle i \rangle] \dashv \vdash C_{\mu \circ \pi_1^{-1}} v. [E(v)\langle i \rangle]$$

C-SURE-PROJ-MANY

$$C_\mu(v, w). [\ulcorner x\langle i \rangle = v(x\langle i \rangle) \urcorner]_{x\langle i \rangle \in X} \dashv \vdash C_{\mu \circ \pi_1^{-1}} v. [\ulcorner x\langle i \rangle = v(x\langle i \rangle) \urcorner]_{x\langle i \rangle \in X}$$

C-EXTRACT

$$C_{\mu_1} v_1. ([E_1\langle i \rangle = v_1] * E_2\langle i \rangle \sim \mu_2) \vdash E_1\langle i \rangle \sim \mu_1 * E_2\langle i \rangle \sim \mu_2$$

C-DIST-PROJ

$$C_\mu(x, y). E\langle i \rangle(x) \sim \mu(x) \vdash C_{\mu \circ \pi_1^{-1}} x. E\langle i \rangle(x) \sim \mu(x)$$

— Relational Lifting —

RL-CONS

$$\frac{R_1 \subseteq R_2}{[R_1] \vdash [R_2]}$$

RL-UNARY

$$\frac{R \subseteq \mathbb{V}^{\{x_1\langle i \rangle, \dots, x_n\langle i \rangle\}}}{[R] \vdash [R(x_1\langle i \rangle, \dots, x_n\langle i \rangle)]}$$

RL-EQ-DIST

$$\frac{i \neq j}{[x\langle i \rangle = y\langle j \rangle] \vdash \exists \mu. x\langle i \rangle \sim \mu * y\langle j \rangle \sim \mu}$$

RL-CONVEX

$$C_\mu _ . [R] \vdash [R]$$

RL-MERGE

$$[R_1] * [R_2] \vdash [R_1 \wedge R_2]$$

RL-SURE-MERGE

$$\frac{R \subseteq \mathbb{V}^X \quad \text{pvar}(e\langle i \rangle) \subseteq X}{[R] * [\ulcorner x\langle i \rangle = e\langle i \rangle \urcorner] \vdash [R \wedge x\langle i \rangle = e\langle i \rangle]}$$

COUPLING

$$\frac{\mu \circ \pi_1^{-1} = \mu_1 \quad \mu \circ \pi_2^{-1} = \mu_2 \quad \mu(R) = 1}{x_1\langle 1 \rangle \sim \mu_1 * x_2\langle 2 \rangle \sim \mu_2 \vdash [R(x_1\langle 1 \rangle, x_2\langle 2 \rangle)]}$$

Fig. 11. Derived rules.

$$\begin{array}{c}
\text{WP-LOOP-0} \\
P \vdash \mathbf{wp} [i: \mathbf{repeat} \ 0 \ t] \{P\}
\end{array}
\quad
\frac{\text{WP-LOOP-LOCKSTEP} \quad \forall k < n. P(k) \vdash \mathbf{wp} [i: t, j: t'] \{P(k+1)\}}{P(0) \vdash \mathbf{wp} [i: (\mathbf{repeat} \ n \ t), j: (\mathbf{repeat} \ n \ t')] \{P(n)\}} \ n \in \mathbb{N}$$

$$\frac{\text{WP-RL-ASSIGN} \quad R \subseteq \mathbb{V}^X \quad x\langle i \rangle \notin \text{pvar}(e\langle i \rangle) \subseteq X \quad \forall y \in \text{pvar}(e). \mathbf{p}(y\langle i \rangle) > 0 \quad \mathbf{p}(x\langle i \rangle) = 1}{[R]@p \vdash \mathbf{wp} [i: x := e] \{[R \wedge x\langle i \rangle = e\langle i \rangle]@p\}}$$

$$\frac{\text{WP-IF-UNARY} \quad P * [e\langle 1 \rangle = 1] \Vdash \mathbf{wp} [1: t_1] \{Q(1)\} \quad P * [e\langle 1 \rangle = 0] \Vdash \mathbf{wp} [1: t_2] \{Q(0)\}}{P * e\langle 1 \rangle \sim \beta \Vdash \mathbf{wp} [1: \mathbf{if} \ e \ \mathbf{then} \ t_1 \ \mathbf{else} \ t_2] \{C_\beta b. Q(b)\}}$$

Fig. 12. Derived WP rules.

over S . Since we are working with countable underlying set throughout, the requirement of μ to be over the full sigma algebra Σ_A is not an extra restriction.

Definition B.2 (Fusion operation). Given $\mu: \mathbb{D}(A)$ and $\kappa: A \rightarrow \mathbb{D}(B)$, we define their fusion $(\mu \prec \kappa): \mathbb{D}(A \times B)$ by:

$$\mu \prec \kappa \triangleq \lambda(v, w). \mu(v) \kappa(v)(w).$$

PROPOSITION B.3. $\mu \prec \kappa = \mathbf{bind}(\mu, \lambda v. (\mathbf{bind}(\kappa(v), \lambda w. \mathbf{return}(v, w))))$.

PROOF. By unfolding the definitions, we obtain

$$\begin{aligned}
& \mathbf{bind}(\mu, \lambda v. (\mathbf{bind}(\kappa(v), \lambda w. \mathbf{return}(v, w))))(a, b) \\
&= \sum_{a' \in A} \mu(a') \cdot \sum_{b' \in B} \kappa(a')(b') \cdot \delta_{(a', b')}(a, b) \\
&= \mu(a) \kappa(a)(b)
\end{aligned}$$

□

LEMMA B.4. For all $\mu: \mathbb{D}(A \times B)$, there exists a $\kappa: A \rightarrow \mathbb{D}(B)$ such that $\mu = (\mu \circ \pi_1^{-1}) \prec \kappa$.

PROOF. Let $\mu_1 = \mu \circ \pi_1^{-1}$. Then the result is immediate by letting $\kappa(a)(b) = \begin{cases} \frac{\mu_0(a, b)}{\mu_1(a)} & \text{if } \mu_1(a) > 0 \\ 0 & \text{otherwise} \end{cases}$. □

B.1 Program Semantics

We assume each primitive operator $\varphi \in \{+, -, <, \dots\}$ has an associated arity $\text{ar}(\varphi) \in \mathbb{N}$, and is given semantics as some function $\llbracket \varphi \rrbracket: \mathbb{V}^{\text{ar}(\varphi)} \rightarrow \mathbb{V}$. Expressions $e \in \mathbb{E}$ are given semantics as a function $\llbracket e \rrbracket: \mathbb{S} \rightarrow \mathbb{V}$ as standard:

$$\llbracket v \rrbracket(s) \triangleq v \quad \llbracket x \rrbracket(s) \triangleq s(x) \quad \llbracket \varphi(e_1, \dots, e_{\text{ar}(\varphi)}) \rrbracket(s) \triangleq \llbracket \varphi \rrbracket(\llbracket e_1 \rrbracket, \dots, \llbracket e_{\text{ar}(\varphi)} \rrbracket)$$

Definition B.5 (Term semantics). Given $t \in \mathbb{T}$ we define its *kernel semantics* $\mathcal{K}\llbracket t \rrbracket: \mathbb{S} \rightarrow \mathbb{D}(\Sigma_{\mathbb{S}})$ as follows:

$$\begin{aligned}
\mathcal{K}\llbracket \mathbf{skip} \rrbracket(s) &\triangleq \mathbf{return}(s) \\
\mathcal{K}\llbracket x := e \rrbracket(s) &\triangleq \mathbf{return}(s[x \mapsto \llbracket e \rrbracket(s)]) \\
\mathcal{K}\llbracket x \approx d(e_1, \dots, e_n) \rrbracket(s) &\triangleq v \leftarrow \llbracket d \rrbracket(\llbracket e_1 \rrbracket(s), \dots, \llbracket e_n \rrbracket(s)); \mathbf{return}(s[x \mapsto v]) \\
\mathcal{K}\llbracket t_1; t_2 \rrbracket(s) &\triangleq s' \leftarrow \mathcal{K}\llbracket t_1 \rrbracket(s); \mathcal{K}\llbracket t_2 \rrbracket(s')
\end{aligned}$$

$$\begin{aligned}\mathcal{K}[\text{if } e \text{ then } t_1 \text{ else } t_2](s) &\triangleq \text{if } \llbracket e \rrbracket(s) \neq 0 \text{ then } \mathcal{K}[\llbracket t_1 \rrbracket](s) \text{ else } \mathcal{K}[\llbracket t_2 \rrbracket](s) \\ \mathcal{K}[\text{repeat } e \text{ } t](s) &\triangleq \text{loop}_t(\llbracket e \rrbracket(s), s)\end{aligned}$$

where loop_t simply iterates t :

$$\text{loop}_t(n, s) \triangleq \begin{cases} \text{return}(s) & \text{if } n \leq 0 \\ s' \leftarrow \text{loop}_t(n-1, s); \mathcal{K}[\llbracket t \rrbracket](s') & \text{otherwise} \end{cases}$$

The semantics of a term is then defined as:

$$\begin{aligned}\llbracket t \rrbracket &: \mathbb{D}(\Sigma_{\mathbb{S}}) \rightarrow \mathbb{D}(\Sigma_{\mathbb{S}}) \\ \llbracket t \rrbracket(\mu) &\triangleq s \leftarrow \mu; \mathcal{K}[\llbracket t \rrbracket](s)\end{aligned}$$

Evaluation contexts \mathcal{E} are defined by the following grammar:

$$\begin{aligned}\mathcal{E} &::= x := \mathcal{E}' \mid x \approx d(\vec{e}_1, \mathcal{E}', \vec{e}_2) \mid \text{if } \mathcal{E}' \text{ then } t_1 \text{ else } t_2 \mid \text{repeat } \mathcal{E}' \text{ } t \\ \mathcal{E}' &::= [\cdot] \mid \varphi(\vec{e}_1, \mathcal{E}', \vec{e}_2)\end{aligned}$$

A simple property holds for evaluation contexts.

LEMMA B.6. $\mathcal{K}[\llbracket \mathcal{E}[e] \rrbracket](s) = \mathcal{K}[\llbracket \mathcal{E}[\llbracket e \rrbracket(s) \rrbracket](s)$.

PROOF. Easy by induction on the structure of evaluation contexts. \square

B.2 Permissions

Section A needs a side-condition on assertions which concerns how an assertion constrains permission ownership. In BLUEBELL, most manipulations do not concern permissions, except for when a mutation takes place, where permissions are used to make sure the frame forgets all information about the variable to be mutated. The notion of *permission-abstract assertion* we now define characterises the assertions which are not chiefly concerned about permissions.

An assertion $P \in \text{HA}_I$ is *permission-abstract* with respect to some $X \subseteq I \times \mathbb{X}$, written $\text{pabs}(P, X)$, if it is invariant under scaling of permission of X ; that is:

$$\text{pabs}(P, X) \triangleq \forall \mathcal{F}, \mu, p, q, n \in \mathbb{N} \setminus \{0\}. P(\mathcal{F}, \mu, p[x\langle i \rangle: q]) \Rightarrow P(\mathcal{F}, \mu, p[x\langle i \rangle: q/n]).$$

For example, fixing $X = \{x\langle i \rangle\}$ then $x\langle i \rangle \sim \mu$, $[x\langle i \rangle = v]$, and $(y\langle i \rangle:1)$ are permission-abstract, but $(x\langle i \rangle: \frac{1}{2})$ is not.

C Measure Theory Lemmas

Notation. In what follows, given $n \in \mathbb{N}$ with $n > 1$, we write $[n]$ to denote the set $\{1, \dots, n\}$. Moreover, for iterated summation we use the notation $\sum_{i \in I(\Phi(i))} f(i)$ where $I = \{i_0, i_1, \dots\}$ is countable and Φ is a predicate on elements of I , to denote the sum $f(j_0) + f(j_1) + \dots$ where j_0, j_1, \dots is the sublist of i_0, i_1, \dots consisting of the elements that satisfy Φ . A similar convention is used for other commutative and associative operators, e.g. \cup . A countable partition of Ω is a partition of Ω , $S \subseteq \mathcal{P}(\Omega)$, with countably many sets. For uniformity, we represent countable partitions as $S = \{A_i\}_{i \in \mathbb{N}}$ with the convention that when the partition has finitely many sets, say n , all the A_i with $i \geq n$ are empty.

As mentioned, BLUEBELL is only concerned with discrete distributions, i.e. distributions over a countable set of outcomes. The following lemma expresses the key property of σ -algebras over countable outcomes that we exploit for proving the other results.

LEMMA C.1. *Let Ω be an countable set, and \mathcal{F} to be an arbitrary σ -algebra on Ω . Then there exists a countable partition S of Ω such that $\mathcal{F} = \sigma(S)$.*

PROOF. For every element $x \in \Omega$, we identify the smallest event $E_x \in \mathcal{F}$ such that $x \in E_x$, and show that for $x, z \in \Omega$, either $E_x = E_z$ or $E_x \cap E_z = \emptyset$. Then the set $S = \{E_x \mid x \in \Omega\}$ is a partition of Ω , and any event $E \in \mathcal{F}$ can be represented as $\bigcup_{x \in E} E_x$, which suffices to show that $\mathcal{F} = \sigma(S)$.

For every x, y , let

$$A_{x,y} = \begin{cases} \Omega & \text{if } \forall E \in \mathcal{F}, \text{ either } x, y \text{ both in } E \text{ or } x, y \text{ both not in } E \\ E & \text{otherwise, pick any } E \in \mathcal{F} \text{ such that } x \in E \text{ and } y \notin E \end{cases}$$

Then we show that, for all x , $E_x = \bigcap_{y \in \Omega} A_{x,y}$ is the smallest event in \mathcal{F} such that $x \in E_x$ as follows. If there exists E'_x such that $x \in E'_x$ and $E'_x \subset E_x$, then $E_x \setminus E'_x$ is not empty. Let y be an element of $E_x \setminus E'_x$, and by the definition of $A_{x,y}$, we have $y \notin A_{x,y}$. Thus, $y \notin \bigcap_{y \in \Omega} A_{x,y} = E_x$, which contradicts with $y \in E_x \setminus E'_x$.

Next, for any $x, z \in \Omega$, since E_x is the smallest event containing x and E_z is the smallest event containing z , the smaller event $E_z \setminus E_x$ is either equivalent to E_z or not containing z .

- If $E_z \setminus E_x = E_z$, then E_x and E_z are disjoint.
- If $z \notin E_z \setminus E_x$, then it must $z \in E_x$, which implies that there exists *no* $E \in \mathcal{F}$ such that $x \in E$ and $z \notin E$. Because \mathcal{F} is closed under complement, then there exists *no* $E \in \mathcal{F}$ such that $x \notin E$ and $z \in E$ as well. Therefore, we have $x \in \bigcap_{y \in \Omega} A_{z,y} = E_z$ as well. Furthermore, because E_z is the smallest event in \mathcal{F} that contains z and E_x also contains z , we have $E_z \subseteq E_x$; symmetrically, we have $E_x \subseteq E_z$. Thus, $E_x = E_z$.

Hence, the set $S = \{E_x \mid x \in \Omega\}$ is a countable partition of Ω . \square

LEMMA C.2. *If $S = \{A_i\}_{i \in \mathbb{N}}$ is a partition of Ω , and $\mathcal{F} = \sigma(S)$, then every event $E \in \mathcal{F}$ can be written as $E = \biguplus_{i \in I} A_i$ for some $I \subseteq \mathbb{N}$. In other words, $\sigma(S) = \{\biguplus_{i \in I} A_i \mid I \subseteq \mathbb{N}\}$.*

PROOF. Because σ -algebras are closed under countable union, for any $I \subseteq \mathbb{N}$, $\biguplus_{i \in I} A_i \in \sigma(S)$. Thus, $\sigma(S) \supseteq \{\biguplus_{i \in I} A_i \mid I \subseteq \mathbb{N}\}$.

Also, $\{\biguplus_{i \in I} A_i \mid I \subseteq \mathbb{N}\}$ is a σ -algebra:

- $\Omega = \biguplus_{i \in \mathbb{N}} A_i$.
- Given a countable sequences of events $E_1 = \biguplus_{i \in I_1} A_i$, $E_2 = \biguplus_{i \in I_2} A_i$, ..., let $I = \bigcup_{j \in \mathbb{N}} I_j$; then we have $\biguplus_{j \in \mathbb{N}} E_j = \biguplus_{i \in I} A_i$.
- If $E = \biguplus_{i \in I} A_i$, then the complement of E is $(\Omega \setminus E) = \biguplus_{i \in (\mathbb{N} \setminus I)} A_i$.

Then, $\{\biguplus_{i \in I} A_i \mid I \subseteq \mathbb{N}\}$ is a σ -algebra that contains S . Therefore, $\sigma(S) = \{\biguplus_{i \in I} A_i \mid I \subseteq \mathbb{N}\}$. \square

LEMMA C.3. *Let Ω be a countable set. If $S_1 = \{A_i\}_{i \in \mathbb{N}}$ and $S_2 = \{B_j\}_{j \in \mathbb{N}}$ are both countable partitions of Ω , then $\sigma(S_1) \subseteq \sigma(S_2)$ implies that for any $B_j \in S_2$ with $B_j \neq \emptyset$, we can find a unique $A_i \in S_1$ such that $B_j \subseteq A_i$.*

PROOF. For any $B_j \in S_2$ with $B_j \neq \emptyset$, pick an arbitrary element $s \in B_j$ and denote the unique element of S_1 that contains s as A_i . Because $A_i \in S_1$ and $S_1 \subset \sigma(S_1) \subseteq \sigma(S_2)$, we have $A_i \in \sigma(S_2)$. Note that $s \in B_j$ and B_j is an element of the partition S_2 that generates $\sigma(S_2)$, B_j must be the smallest event in $\sigma(S_2)$ that contains s . Because $s \in A_i$ as well, B_j being the smallest event containing s implies that $B_j \subseteq A_i$. \square

LEMMA C.4. *Assume we are given a σ -algebra \mathcal{F}_1 over a countable set Ω , measure $\mu_1 \in \mathbb{D}(\mathcal{F}_1)$, a countable set A , a distribution $\mu \in \Sigma_A$, and a function $\kappa_1: A \rightarrow \mathbb{D}(\mathcal{F}_1)$ such that $\mu_1 = \mathbf{bind}(\mu, \kappa_1)$. Then, for any probability space (\mathcal{F}_2, μ_2) such that $(\mathcal{F}_1, \mu_1) \sqsubseteq (\mathcal{F}_2, \mu_2)$, there exists κ_2 such that $\mu_2 = \mathbf{bind}(\mu, \kappa_2)$, and that for any $a \in \text{supp}(\mu)$, $(\mathcal{F}_1, \kappa_1(a)) \sqsubseteq (\mathcal{F}_2, \kappa_2(a))$.*

PROOF. By Lemma C.1, $\mathcal{F}_i = \sigma(S_i)$ for some countable partition S_i . Also, $(\mathcal{F}_1, \mu_1) \sqsubseteq (\mathcal{F}_2, \mu_2)$ implies that $\mathcal{F}_1 \subseteq \mathcal{F}_2$. So we have $\sigma(S_1) \subseteq \sigma(S_2)$, which by Lemma C.3 implies that for any $B \in S_2$ with $B \neq \emptyset$, we can find a unique $A \in S_1$ such that $B \subseteq A$. Let f be the mapping associating to any $B \neq \emptyset$ the corresponding $A = f(B)$, and $f(B) = \emptyset$ when $B = \emptyset$.

Then, we define κ_2 as follows: for any $a \in A$, $E \in \mathcal{F}_2$, there exists $S \subseteq S_2$ such that $E = \bigsqcup_{B \in S} B$, then define

$$\kappa_2(a)(E) = \sum_{B \in S} \kappa_1(a)(f(B)) \cdot h(B),$$

where $h(B) = \mu_2(B)/\mu_2(f(B))$ if $\mu_2(f(B)) \neq 0$ and $h(B) = 0$ otherwise.

Then we calculate:

$$\begin{aligned} & \mathbf{bind}(\mu, \kappa_2)(E) \\ &= \sum_{a \in A} \mu(a) \cdot \kappa_2(E) \\ &= \sum_{a \in A} \mu(a) \cdot \sum_{B \in S} \kappa_1(a)(f(B)) \cdot h(B) \\ &= \sum_{B \in S} \sum_{a \in A} \mu(a) \cdot \kappa_1(a)(f(B)) \cdot h(B) \\ &= \sum_{B \in S} \mathbf{bind}(\mu, \kappa_1)(f(B)) \cdot h(B) \\ &= \sum_{B \in S} \mu_1(f(B)) \cdot h(B) \\ &= \sum_{B \in S | \mu_2(f(B)) \neq 0} \mu_1(f(B)) \cdot \frac{\mu_2(B)}{\mu_2(f(B))} \\ &= \sum_{B \in S | \mu_2(f(B)) \neq 0} \mu_2(f(B)) \cdot \frac{\mu_2(B)}{\mu_2(f(B))} && (\mu_1(E') = \mu_2(E') \text{ for any } E' \in \mathcal{F}_1) \\ &= \sum_{B \in S | \mu_2(f(B)) \neq 0} \mu_2(B) \\ &= \sum_{B \in S | \mu_2(f(B)) \neq 0} \mu_2(B) + \sum_{B \in S | \mu_2(f(B)) = 0} \mu_2(B) && (\text{Because } \mu_2(f(B)) = 0 \text{ implies } \mu_2(B) = 0) \\ &= \sum_{B \in S} \mu_2(B) \\ &= \mu_2(\bigsqcup_{B \in S} B) \\ &= \mu_2(E) \end{aligned}$$

Thus, $\mathbf{bind}(\mu, \kappa_2) = \mu_2$.

Also, for any $a \in A_\mu$, for any $E \in \mathcal{F}_1$, there exists $S' \subseteq S_1$ such that $E = \bigsqcup_{A \in S'} A$.

$$\begin{aligned} \kappa_2(a)(E) &= \kappa_2(a)(\bigsqcup_{A \in S'} A) \\ &= \sum_{A \in S'} \kappa_2(a)(A) \\ &= \sum_{A \in S'} \sum_{B \subseteq A | B \in \mathcal{F}_2} \kappa_2(a)(B) \end{aligned}$$

$$\begin{aligned}
&= \sum_{A \in S'} \sum_{B \subseteq A \mid B \in \mathcal{F}_2, \mu_2(f(B)) \neq 0} \kappa_1(a)(f(B)) \cdot \frac{\mu_2(B)}{\mu_2(f(B))} \\
&= \sum_{A \in S' \mid \mu_2(A) \neq 0} \kappa_1(a)(A) \cdot \frac{(\sum_{B \subseteq A \mid B \in \mathcal{F}_2} \mu_2(B))}{\mu_2(A)} \\
&= \sum_{A \in S' \mid \mu_2(A) \neq 0} \kappa_1(a)(A) \cdot \frac{\mu_2(A)}{\mu_2(A)} \\
&= \sum_{A \in S' \mid \mu_2(A) \neq 0} \kappa_1(a)(A) \\
&= \sum_{A \in S'} \kappa_1(a)(A) \\
&= \kappa_1(a)(\biguplus_{A \in S'} A) \\
&= \kappa_1(a)(E)
\end{aligned}$$

Thus, for any a , $(\sigma_1, \kappa_1(a)) \sqsubseteq (\sigma_2, \kappa_2(a))$. \square

LEMMA C.5. *Given two σ -algebras \mathcal{F}_1 and \mathcal{F}_2 over two countable underlying sets Ω_1, Ω_2 , then a general element in the product σ -algebra $\mathcal{F}_1 \otimes \mathcal{F}_2$ can be expressed as $\biguplus_{(i,j) \in I} (A_i \times B_j)$ for some $I \subseteq \mathbb{N}^2$ and $A_i \in \mathcal{F}_1, B_j \in \mathcal{F}_2$ for $(i, j) \in I$.*

PROOF. By Lemma C.1, each σ -algebra \mathcal{F}_i is generated by a countable partition over Ω_i . Let $S_1 = \{A_i\}_{i \in \mathbb{N}}$ be the countable partition that generates \mathcal{F}_1 , $S_2 = \{B_i\}_{i \in \mathbb{N}}$ be the countable partition that generates \mathcal{F}_2 . By Lemma C.2, a general element in \mathcal{F}_1 can be written as $\biguplus_{j \in J} A_j$ for some $J \subseteq \mathbb{N}$, and similarly, a general element in \mathcal{F}_2 can be written as $\biguplus_{k \in K} B_k$ for some $K \subseteq \mathbb{N}$.

Note that $\{A_j \times B_k\}_{j,k \in \mathbb{N}}$ is a partition because: if $(A_j \times B_k) \cap (A_{j'} \times B_{k'}) \neq \emptyset$ for some $j \neq j'$ and $k \neq k'$, then it must $A_j \cap A_{j'} \neq \emptyset$ and $B_k \cap B_{k'} \neq \emptyset$, and that imply that $A_j = A_{j'}$ and $B_j = B_{j'}$; therefore, $A_j \times B_k = A_{j'} \times B_{k'}$.

We next show that $\mathcal{F}_1 \otimes \mathcal{F}_2$ is generated by the partition $\{A_j \times B_k\}_{j,k \in \mathbb{N}}$.

$$\begin{aligned}
\mathcal{F}_1 \otimes \mathcal{F}_2 &= \sigma(\mathcal{F}_1 \times \mathcal{F}_2) \\
&= \sigma(\{\biguplus_{j \in J_1} A_j \times \biguplus_{j \in J_2} B_j \mid J_1, J_2 \subseteq \mathbb{N}\}) \\
&= \sigma(\{\biguplus_{j \in J_1, k \in J_2} (A_j \times B_k) \mid J_1, J_2 \subseteq \mathbb{N}\}) \\
&= \sigma(\{A_j \times B_k \mid j, k \subseteq \mathbb{N}\})
\end{aligned}$$

Since each $A_j \in S_1 \subseteq \mathcal{F}_1$ and $B_k \in S_2 \subseteq \mathcal{F}_2$ a general element in $\mathcal{F}_1 \otimes \mathcal{F}_2$ can be expressed as $\{\biguplus_{j,k \subseteq I} (A_j \times B_k) \mid A_j \in \mathcal{F}_1, B_k \in \mathcal{F}_2, I \subseteq \mathbb{N}^2\}$ according to Lemma C.1. \square

LEMMA C.6. *Given two probability spaces $(\mathcal{F}_a, \mu_a), (\mathcal{F}_b, \mu_b) \in \mathbb{P}(\Omega)$, their independent product $(\mathcal{F}_a, \mu_a) \otimes (\mathcal{F}_b, \mu_b)$ exists if $\mu_a(E_a) \cdot \mu_b(E_b) = 0$ for any $E_a \in \mathcal{F}_a, E_b \in \mathcal{F}_b$ such that $E_a \cap E_b = \emptyset$.*

PROOF. We first define $\mu : \{E_a \cap E_b \mid E_a \in \mathcal{F}_a, E_b \in \mathcal{F}_b\} \rightarrow [0, 1]$ by $\mu(E_a \cap E_b) = \mu_a(E_a) \cdot \mu_b(E_b)$ for any $E_a \in \mathcal{F}_a, E_b \in \mathcal{F}_b$, and then show that μ could be extended to a probability measure on $\mathcal{F}_a \oplus \mathcal{F}_b$.

- We first need to show that μ is **well-defined**. That is, $E_a \cap E_b = E'_a \cap E'_b$ implies $\mu_a(E_a) \cdot \mu_b(E_b) = \mu_a(E'_a) \cdot \mu_b(E'_b)$.
When $E_a \cap E_b = E'_a \cap E'_b$, it must $E_a \cap E'_a \supseteq E_a \cap E_b = E'_a \cap E'_b$. Thus, $E_a \setminus E'_a \subseteq E_a \setminus E_b$, and then $E_a \setminus E'_a$ is disjoint from E_b ; symmetrically, $E'_a \setminus E_a$ is disjoint from E'_b . Since E_a, E'_a are both in \mathcal{F}_a , we have $E_a \setminus E'_a$ and $E'_a \setminus E_a$ both measurable in \mathcal{F}_a . Their disjointness and the

result above implies that $\mu_a(E_a \setminus E'_a) \cdot \mu_b(E_b) = 0$ and $\mu_a(E'_a \setminus E_a) \cdot \mu_b(E'_b) = 0$. Symmetric reasoning can also show that $E'_b \setminus E_b$ is disjoint from $E'_a \cap E_a$, and $E_b \setminus E'_b$ is disjoint from $E'_a \cap E_a$, which implies $\mu_a(E_b \setminus E'_b) \cdot \mu_b(E'_a \cap E_a) = 0$ and $\mu_a(E'_b \setminus E_b) \cdot \mu_b(E'_a) = 0$.

Then there are four possibilities:

- If $\mu_b(E_b) = 0$ and $\mu_b(E'_b) = 0$, then $\mu_a(E_a) \cdot \mu_b(E_b) = 0 = \mu_a(E'_a) \cdot \mu_b(E'_b)$.
- If $\mu_a(E_a \setminus E'_a) = 0$ and $\mu_b(E'_a \setminus E_a) = 0$. Then

$$\begin{aligned} \mu_a(E_a) \cdot \mu_b(E_b) &= \mu_a((E'_a \setminus E_a) \uplus (E'_a \cap E_a)) \cdot \mu_b(E_b) \\ &= (\mu_a(E'_a \setminus E_a) + \mu_a(E'_a \cap E_a)) \cdot \mu_b(E_b) \\ &= \mu_a(E'_a \cap E_a) \cdot \mu_b(E_b) \\ &= (\mu_a(E_a \setminus E'_a) + \mu_a(E'_a \cap E_a)) \cdot \mu_b(E_b) \\ &= \mu_a(E'_a) \cdot \mu_b(E_b) \end{aligned}$$

Thus, either $\mu_a(E'_a \cap E_a) = 0$, which implies that

$$\mu_a(E_a) \cdot \mu_b(E_b) = (0 + 0) \cdot \mu_b(E_b) = 0 = (0 + 0) \cdot \mu_b(E_b) = \mu_a(E'_a) \cdot \mu_b(E'_b),$$

or we have both $\mu_b(E'_b \setminus E_b) = 0$ and $\mu_b(E_b \setminus E'_b) = 0$, which imply that

$$\begin{aligned} \mu_a(E_a) \cdot \mu_b(E_b) &= \mu_a(E'_a) \cdot \mu_b(E_b) \\ &= \mu_a(E'_a) \cdot \mu_b((E_b \cap E'_b) \uplus (E_b \setminus E'_b)) \\ &= \mu_a(E'_a) \cdot (\mu_b(E_b \cap E'_b) + 0) \\ &= \mu_a(E'_a) \cdot (\mu_b(E_b \cap E'_b) + \mu_b(E'_b \setminus E_b)) \\ &= \mu_a(E'_a) \cdot \mu_b(E'_b). \end{aligned}$$

- If $\mu_b(E'_b) = 0$ and $\mu_b(E_a \setminus E'_a) = 0$, then

$$\begin{aligned} \mu_a(E_a) \cdot \mu_b(E_b) &= (\mu_a(E_a \cap E'_a) + \mu_a(E_a \setminus E'_a)) \cdot (\mu_b(E_b \cap E'_b) + \mu_b(E_b \setminus E'_b)) \\ &= \mu_a(E_a \cap E'_a) \cdot \mu_b(E_b \setminus E'_b) \end{aligned}$$

Because $\mu_a(E_b \setminus E'_b) \cdot \mu_b(E'_a \cap E_a) = 0$ and $\mu_a(E'_b \setminus E_b) \cdot \mu_b(E'_a) = 0$. Thus, $\mu_a(E_a) \cdot \mu_b(E_b) = 0 = \mu_a(E'_a) \cdot \mu_b(E'_b)$.

- If $\mu_b(E_b) = 0$ and $\mu_b(E'_a \setminus E_a) = 0$, then symmetric as above.

In all these cases, $\mu_a(E_a) \cdot \mu_b(E_b) = \mu_a(E'_a) \cdot \mu_b(E'_b)$ as desired.

- Show that μ satisfy **countable additivity** in $\{E_a \cap E_b \mid E_a \in \mathcal{F}_a, E_b \in \mathcal{F}_b\}$.

We start with showing that μ is finite-additive. Suppose $E_a^n \cap E_b^n = \biguplus_{i \in [n]} (A_i \cap B_i)$ where each $A_i \in \mathcal{F}_a$ and $B_i \in \mathcal{F}_b$. Fix any $A_i \cap B_i$, there is unique minimal $A \in \mathcal{F}_a$ containing $A_i \cap B_i$, because if $A \supseteq A_i \cap B_i$ and $A' \supseteq A_i \cap B_i$, then $A \cap A' \supseteq A_i \cap B_i$ and $A \cap A' \in \mathcal{F}_a$ too, and $A \cap A'$ is smaller. Because we have shown that μ is well-defined, in the following proof, we can assume without loss of generality that A_i is the smallest set in \mathcal{F}_a containing $A_i \cap B_i$. Similarly, we let B_i to be the smallest set in \mathcal{F}_b containing $A_i \cap B_i$. Thus, $E_a^n \cap E_b^n = \biguplus_{i \in [n]} (A_i \cap B_i)$ implies every A_i is smaller than E_a^n and every B_i is smaller than E_b^n . Therefore, $E_a^n \supseteq \bigcup_{i \in [n]} A_i$ and $E_b^n \supseteq \bigcup_{i \in [n]} B_i$, which implies that

$$E_a^n \cap E_b^n \supseteq (\bigcup_{i \in [n]} A_i) \cap (\bigcup_{i \in [n]} B_i) \supseteq \bigcup_{i \in [n]} (A_i \cap B_i) = E_a^n \cap E_b^n,$$

which implies that the \supseteq in the inequalities all collapse to $=$.

For any $I \subseteq [n]$, define $\alpha_I = \bigcap_{i \in I} A_i \setminus (\bigcup_{i \in [n] \setminus I} A_i)$, and $\beta_I = \bigcap_{i \in I} B_i \setminus (\bigcup_{i \in [n] \setminus I} B_i)$. For any $I \neq I'$, $\alpha_I \cap \alpha_{I'} = \emptyset$. Thus, $\{\alpha_I\}_{I \subseteq [n]}$ is a set of disjoint sets in $\bigcup_{i \in [n]} A_i$, and similarly,

$\{\beta_I\}_{I \subseteq [n]}$ is a set of disjoint sets in $\cup_{i \in [n]} B_i$. Also, for any $i \in [n]$, we have $A_i = \cup_{I \subseteq [n] | i \in I} \alpha_I$ and $B_i = \cup_{I \subseteq [n] | i \in I} \beta_I$. Furthermore, for any I ,

$$\alpha_I \cap \cup_{i \in [n]} B_i \subseteq (\cup_{i \in [n]} A_i) \cap (\cup_{i \in [n]} B_i) = \uplus_{i \in [n]} A_i \cap B_i,$$

and thus,

$$\begin{aligned} \alpha_I \cap \cup_{i \in [n]} B_i &= (\uplus_{i \in [n]} A_i \cap B_i) \cap (\alpha_I \cap \cup_{i \in [n]} B_i) \\ &= \uplus_{i \in [n]} (A_i \cap B_i \cap \alpha_I \cap \cup_{j \in [n]} B_j) \\ &= \uplus_{i \in I} (A_i \cap B_i \cap \alpha_I \cap \cup_{j \in [n]} B_j) && (A_i \cap \alpha_I = \emptyset \text{ if } i \notin I) \\ &= \uplus_{i \in I} (A_i \cap B_i \cap \alpha_I) && (B_i \cap \cup_{j \in [n]} B_j = B_i \text{ for any } i) \\ &= \uplus_{i \in I} (B_i \cap \alpha_I) && (A_i \cap \alpha_I = \alpha_I \text{ for any } i \in I) \\ &= \alpha_I \cap \cup_{i \in I} B_i && (8) \end{aligned}$$

Now,

$$\begin{aligned} &\mu(E_a^n \cap E_b^n) \\ &= \mu((\cup_{i \in [n]} A_i) \cap (\cup_{i \in [n]} B_i)) \\ &= \mu((\uplus_{I \subseteq [n]} \alpha_I) \cap (\cup_{i \in [n]} B_i)) && (\text{By definition of } \alpha_I) \\ &= \mu_a(\uplus_{I \subseteq [n]} \alpha_I) \cdot \mu_b(\cup_{i \in [n]} B_i) && (\text{By definition of } \mu) \\ &= \left(\sum_{I \subseteq [n]} \mu_a(\alpha_I) \right) \cdot \mu_b(\cup_{i \in [n]} B_i) && (\text{By finite-additivity of } \mu_a) \\ &= \sum_{I \subseteq [n]} \mu_a(\alpha_I) \cdot \mu_b(\cup_{i \in [n]} B_i) \\ &= \sum_{I \subseteq [n]} \mu(\alpha_I \cap (\cup_{i \in [n]} B_i)) && (\text{By definition of } \mu) \\ &= \sum_{I \subseteq [n]} \mu(\alpha_I \cap (\cup_{i \in I} B_i)) && (\text{By Eq. (8)}) \\ &= \sum_{I \subseteq [n]} \mu_a(\alpha_I) \cdot \mu_b(\cup_{i \in I} B_i) && (\text{By definition of } \mu) \\ &= \sum_{I \subseteq [n]} \mu_a(\alpha_I) \cdot \mu_b(\cup_{i \in I} (\uplus_{I' \subseteq [n] | i \in I'} \beta_{I'})) && (\text{By definition of } \beta_I) \\ &= \sum_{I \subseteq [n]} \mu_a(\alpha_I) \cdot \mu_b(\uplus_{I' \subseteq [n] | I \cap I' \neq \emptyset} \beta_{I'}) \\ &= \sum_{I \subseteq [n]} \mu_a(\alpha_I) \cdot \sum_{I' \subseteq [n] | I \cap I' \neq \emptyset} \mu_b(\beta_{I'}) \\ &= \sum_{I \subseteq [n]} \sum_{I' \subseteq [n] | I \cap I' \neq \emptyset} \mu_a(\alpha_I) \cdot \mu_b(\beta_{I'}) \end{aligned}$$

Meanwhile, for any I, I' , if $|I \cap I'| \geq 2$, then there exists some j, k such that $j \in I \cap I'$ and $k \in I \cap I'$, so

$$\begin{aligned} \mu_a(\alpha_I) \cdot \mu_b(\beta_{I'}) &= \mu_a(\cap_{i \in I} A_i \setminus (\cup_{i \in [n] \setminus I} A_i)) \cdot \mu_b(\cap_{i \in I} B_i \setminus (\cup_{i \in [n] \setminus I} B_i)) \\ &\leq \mu_a(A_j \cap A_k) \cdot \mu_b(B_j \cap B_k) \\ &= \mu(A_j \cap A_k \cap B_j \cap B_k) \end{aligned}$$

$$\begin{aligned}
&= \mu((A_j \cap B_j) \cap (A_k \cap B_k)) \\
&= \mu(\emptyset) \\
&= 0.
\end{aligned}$$

Thus, continuing our previous derivation,

$$\begin{aligned}
&\mu(E_a^n \cap E_b^n) \\
&= \sum_{I \subseteq [n]} \sum_{I' \subseteq [n] \mid I \cap I' \neq \emptyset} \mu_a(\alpha_I) \cdot \mu_b(\beta_{I'}) \\
&= \sum_{I \subseteq [n]} \sum_{I' \subseteq [n] \mid 1=|I \cap I'|} \mu_a(\alpha_I) \cdot \mu_b(\beta_{I'}) && \text{(Because } \mu_a(\alpha_I) \cdot \mu_b(\beta_{I'}) = 0 \text{ if } |I \cap I'| \geq 2) \\
&= \sum_{i \in [n]} \sum_{I \subseteq [n] \mid i \in I} \sum_{I' \subseteq [n] \mid I \cap I' = \{i\}} \mu_a(\alpha_I) \cdot \mu_b(\beta_{I'}) \\
&= \sum_{i \in [n]} \sum_{I \subseteq [n] \mid i \in I} \sum_{I' \subseteq [n] \mid i \in I'} \mu_a(\alpha_I) \cdot \mu_b(\beta_{I'}) && \text{(Because } \mu_a(\alpha_I) \cdot \mu_b(\beta_{I'}) = 0 \text{ if } |I \cap I'| \geq 2) \\
&= \sum_{i \in [n]} \left(\sum_{I \subseteq [n] \mid i \in I} \mu_a(\alpha_I) \cdot \sum_{I' \subseteq [n] \mid i \in I'} \mu_b(\beta_{I'}) \right) \\
&= \sum_{i \in [n]} \mu_a(A_i) \cdot \mu_b(B_i) \\
&= \sum_{i \in [n]} \mu(A_i \cap B_i)
\end{aligned}$$

Thus, we established the finite additivity. For countable additivity, suppose $E_a \cap E_b = \biguplus_{i \in \mathbb{N}} (A_i \cap B_i)$. By the same reason as above, we also have

$$E_a \cap E_b = (\cup_{i \in \mathbb{N}} A_i) \cap (\cup_{i \in \mathbb{N}} B_i) = \cup_{i \in \mathbb{N}} (A_i \cap B_i) = E_a \cap E_b.$$

Then,

$$\begin{aligned}
&\mu(E_a \cap E_b) \\
&= \mu((\cup_{i \in \mathbb{N}} A_i) \cap (\cup_{i \in \mathbb{N}} B_i)) \\
&= \mu_a(\cup_{i \in \mathbb{N}} A_i) \cdot \mu_b(\cup_{i \in \mathbb{N}} B_i) \\
&= \mu_a(\lim_{n \rightarrow \infty} \cup_{i \in [n]} A_i) \cdot \mu_b(\lim_{n \rightarrow \infty} \cup_{i \in [n]} B_i) \\
&= \lim_{n \rightarrow \infty} \mu_a(\cup_{i \in [n]} A_i) \cdot \lim_{n \rightarrow \infty} \mu_b(\cup_{i \in [n]} B_i) && \text{(By continuity of } \mu_a \text{ and } \mu_b) \\
&= \lim_{n \rightarrow \infty} \mu_a(\cup_{i \in [n]} A_i) \cdot \mu_b(\cup_{i \in [n]} B_i) && (\dagger) \\
&= \lim_{n \rightarrow \infty} \sum_{i \in [n]} \mu_b(B_i) \cdot \mu_a(A_i) && \text{(By Eq. (8))} \\
&= \sum_{i \in \mathbb{N}} \mu_b(B_i) \cdot \mu_a(A_i), && (9)
\end{aligned}$$

where (\dagger) holds because that the product of limits equals to the limit of the product when both $\lim_{n \rightarrow \infty} \mu_a(\cup_{i \in [n]} A_i)$ and $\lim_{n \rightarrow \infty} \mu_b(\cup_{i \in [n]} B_i)$ are finite. Thus, we proved countable additivity as well.

- Next we show that we can **extend μ to a measure on $\mathcal{F}_a \oplus \mathcal{F}_b$** .

So far, we proved that μ is a sub-additive measure on the $\{E_a \cap E_b \mid E_a \in \mathcal{F}_a, E_b \in \mathcal{F}_b\}$, which forms a π -system. By a known theorem in probability theory (e.g. [Rosenthal 2006, Corollary 2.5.4]), we can extend a sub-additive measure on a π -system to the σ -algebra it generates if the π -system is a semi-algebra. Thus, we can extend μ to a measure on $\sigma(\{E_a \cap E_b \mid E_a \in \mathcal{F}_a, E_b \in \mathcal{F}_b\})$ if we can prove $J = \{E_a \cap E_b \mid E_a \in \mathcal{F}_a, E_b \in \mathcal{F}_b\}$ is a semi-algebra.

- J contains \emptyset and Ω : trivial.
- J is closed under finite intersection: $(E_a \cap E_b) \cap (E'_a \cap E'_b) = (E_a \cap E'_a) \cap (E_b \cap E'_b)$, where $E_a \cap E'_a \in \mathcal{F}_a$, and $E_b \cap E'_b \in \mathcal{F}_b$.
- The complement of any element of J is equal to a finite disjoint union of elements of J :

$$\begin{aligned} (E_a \cap E_b)^C &= E_a^C \cup E_b^C \\ &= (E_a^C \cap \Omega) \uplus (E_a \cap E_b^C) \end{aligned}$$

where $E_a^C, E_a \in \mathcal{F}_a$, and $E_b^C, \Omega \in \mathcal{F}_b$.

As shown in [Li et al. 2023a],

$$\sigma(\{E_a \cap E_b \mid E_a \in \mathcal{F}_a, E_b \in \mathcal{F}_b\}) = \mathcal{F}_a \oplus \mathcal{F}_b \quad (10)$$

Thus, the extension of μ is a measure on $\mathcal{F}_a \oplus \mathcal{F}_b$.

- Last, we show that μ is a **probability measure** on $\mathcal{F}_a \oplus \mathcal{F}_b$: $\mu(\Omega) = \mu_a(\Omega) \cdot \mu_b(\Omega) = 1$. \square

LEMMA C.7. Consider two probability spaces $(\mathcal{F}_1, \mu_1), (\mathcal{F}_2, \mu_2) \in \mathbb{P}(\Omega)$, and some other probability space (Σ_A, μ) and kernel κ such that $\mu_1 = \mathbf{bind}(\mu, \kappa)$.

Then, the independent product $(\mathcal{F}_1, \mu_1) \otimes (\mathcal{F}_2, \mu_2)$ exists if and only if for any $a \in \text{supp}(\mu)$, the independent product $(\mathcal{F}_1, \kappa(a)) \otimes (\mathcal{F}_2, \mu_2)$ exists. When they both exist,

$$(\mathcal{F}_1, \mu_1) \otimes (\mathcal{F}_2, \mu_2) = (\mathcal{F}_1 \oplus \mathcal{F}_2, \mathbf{bind}(\mu, \lambda a. \kappa(a) \otimes \mu_2))$$

PROOF. We first show the backwards direction. By Lemma C.6, for any $a \in \text{supp}(\mu)$, to show that the independent product $(\mathcal{F}_1, \kappa(a)) \otimes (\mathcal{F}_1, \mu_1)$ exists, it suffices to show that for any $E_1 \in \mathcal{F}_1, E_2 \in \mathcal{F}_2$ such that $E_1 \cap E_2 = \emptyset$, $\kappa(a)(E_1) \cdot \mu_2(E_2) = 0$.

Fix any such E_1, E_2 , because $(\mathcal{F}_1, \mu_1) \otimes (\mathcal{F}_2, \mu_2)$ is defined, we have $\mu_1(E_1) \cdot \mu_2(E_2) = 0$, then either $\mu_1(E_1) = 0$ or $\mu_2(E_2) = 0$.

- If $\mu_1(E_1) = 0$: Recall that

$$\mu_1(E_1) = \mathbf{bind}(\mu, \kappa)(E_1) = \sum_{a \in A} \mu(a) \cdot \kappa(a)(E_1) = \sum_{a \in \text{supp}(\mu)} \mu(a) \cdot \kappa(a)(E_1)$$

Because all $\mu(a) > 0$ and $\kappa(a)(E_1) \geq 0$ for all $a \in \text{supp}(\mu)$, $\sum_{a \in \text{supp}(\mu)} \mu(a) \cdot \kappa(a)(E_1) = 0$ implies that $\mu(a) \cdot \kappa(a)(E_1) = 0$ for all $a \in \text{supp}(\mu)$. Thus, for all $a \in \text{supp}(\mu)$, it must $\kappa(a)(E_1) = 0$. Therefore, $\kappa(a)(E_1) \cdot \mu_2(E_2) = 0$ for all $a \in \text{supp}(\mu)$ with this E_1, E_2 .

- If $\mu_2(E_2) = 0$, then it is also clear that $\kappa(a)(E_1) \cdot \mu_2(E_2) = 0$ for all $a \in \text{supp}(\mu)$.

Thus, we have $\kappa(a)(E_1) \cdot \mu_2(E_2) = 0$ for any $E_1 \cap E_2 = \emptyset$ and $a \in \text{supp}(\mu)$. By Lemma C.6, the independent product $(\mathcal{F}_1, \kappa(a)) \otimes (\mathcal{F}_1, \mu_1)$ exists.

For the forward direction: for any $E_1 \in \mathcal{F}_1$ and $E_2 \in \mathcal{F}_2$ such that $E_1 \cap E_2 = \emptyset$, the independent product $(\mathcal{F}_1, \kappa(a)) \otimes (\mathcal{F}_2, \mu_2)$ exists implies that

$$\kappa(a)(E_1) \cdot \mu_2(E_2) = 0.$$

Thus,

$$\mu_1(E_1) \cdot \mu_2(E_2) = \mathbf{bind}(\mu, \kappa)(E_1) \cdot \mu_2(E_2)$$

$$\begin{aligned}
&= \left(\sum_{a \in A} \mu(a) \cdot \kappa(a)(E_1) \right) \cdot \mu_2(E_2) \\
&= \sum_{a \in A_\mu} \mu(a) \cdot (\kappa(a)(E_1) \cdot \mu_2(E_2)) \\
&= \sum_{a \in A_\mu} \mu(a) \cdot 0 = 0
\end{aligned}$$

Thus, by Lemma C.6, the independent product $(\mathcal{F}_1, \mu_1) \otimes (\mathcal{F}_2, \mu_2)$ exists. For any $E_1 \in \mathcal{F}_1$ and $E_2 \in \mathcal{F}_2$,

$$\begin{aligned}
&\mathbf{bind}(\mu, \lambda a. \kappa(a) \otimes \mu_2)(E_1 \cap E_2) \\
&= \sum_{a \in \text{supp}(\mu)} \mu(a) \cdot (\kappa(a) \otimes \mu_2)(E_1 \cap E_2) \\
&= \sum_{a \in \text{supp}(\mu)} \mu(a) \cdot \kappa(a)(E_1) \cdot \mu_2(E_2) \\
&= \left(\sum_{a \in \text{supp}(\mu)} \mu(a) \cdot \kappa(a)(E_1) \right) \cdot \mu_2(E_2) \\
&= \mathbf{bind}(\mu, \kappa)(E_1) \cdot \mu_2(E_2) \\
&= \mu_1(E_1) \cdot \mu_2(E_2) \\
&= (\mu_1 \otimes \mu_2)(E_1 \cap E_2)
\end{aligned}$$

Thus, $(\mathcal{F}_1, \mu_1) \otimes (\mathcal{F}_2, \mu_2) = (\mathcal{F}_1 \oplus \mathcal{F}_2, \mathbf{bind}(\mu, \lambda a. \kappa(a) \otimes \mu_2))$. \square

D Model

D.1 Basic Connectives

The following are the definitions of the standard SL connectives we use in BLUEBELL:

$$\begin{aligned}
\lceil \varphi \rceil &\triangleq \lambda _ . \varphi & P * Q &\triangleq \lambda a. \exists b_1, b_2. (b_1 \cdot b_2) \preceq a \wedge P(b_1) \wedge Q(b_2) \\
\text{Own}(b) &\triangleq \lambda a. b \preceq a & P \multimap Q &\triangleq \lambda a. \forall b. \mathcal{V}(a \cdot b) \Rightarrow P(b) \Rightarrow Q(a \cdot b) \\
P \wedge Q &\triangleq \lambda a. P(a) \wedge Q(a) & \forall x : X. P(x) &\triangleq \lambda a. \forall x \in X. P(x)(a) \\
P \vee Q &\triangleq \lambda a. P(a) \vee Q(a) & \exists x : X. P(x) &\triangleq \lambda a. \exists x \in X. P(x)(a)
\end{aligned}$$

D.2 Construction of the BLUEBELL Model

LEMMA D.1. *The structure PSp is an ordered unital resource algebra (RA) as defined in Definition 4.1.*

PROOF. We defined \cdot and \preceq the same way as in [Li et al. 2023a], and they have proved that \cdot is associative and commutative, and \preceq is transitive and reflexive. We check the rest of conditions one by one.

- **Condition** $a \cdot b = b \cdot a$. The independent product is proved to be commutative in [Li et al. 2023a].
- **Condition** $(a \cdot b) \cdot c = a \cdot (b \cdot c)$. The independent product is proved to be associative in [Li et al. 2023a].
- **Condition** $a \preceq b \Rightarrow b \preceq c \Rightarrow a \preceq c$. The order \preceq is proved to be transitive in [Li et al. 2023a].
- **Condition** $a \preceq a$. The order \preceq is proved to be reflexive in [Li et al. 2023a].
- **Condition** $\mathcal{V}(a \cdot b) \Rightarrow \mathcal{V}(a)$. Pattern matching on $a \cdot b$, either there exists probability spaces $\mathcal{P}_1, \mathcal{P}_2$ such that $a = \mathcal{P}_1$, $b = \mathcal{P}_2$ and $\mathcal{P}_1 \otimes \mathcal{P}_2$ is defined, or $a \cdot b = \frac{1}{2}$.

- **Case** $a \cdot b = \frac{1}{2}$. Note that $\mathcal{V}(a \cdot b)$ does not hold when $a \cdot b = \frac{1}{2}$, so we can eliminate this case by ex falso quodlibet.
- **Case** $a \cdot b = \mathcal{P}_1 \otimes \mathcal{P}_2$. Then $a = \mathcal{P}_1$, and thus $\mathcal{V}(a)$.
- **Condition** $\mathcal{V}(\varepsilon)$. Clear because $\varepsilon \neq \frac{1}{2}$.
- **Condition** $a \leq b \Rightarrow \mathcal{V}(b) \Rightarrow \mathcal{V}(a)$. Pattern matching on a and b , either there exists probability spaces $\mathcal{P}_1, \mathcal{P}_2$ such that $a = \mathcal{P}_1$, $b = \mathcal{P}_2$ and $\mathcal{P}_1 \sqsubseteq \mathcal{P}_2$ is defined, or $b = \frac{1}{2}$.
 - **Case** $b = \frac{1}{2}$. Then $\mathcal{V}(b)$ does not hold, and we can eliminate this case by ex falso quodlibet.
 - **Case** $a = \mathcal{P}_1$, $b = \mathcal{P}_2$ and $\mathcal{P}_1 \sqsubseteq \mathcal{P}_2$. We clearly have $\mathcal{V}(a)$.
- **Condition** $\varepsilon \cdot a = a$. Pattern matching on a , either $a = \frac{1}{2}$ or there exists some probability space \mathcal{P} such that $a = \mathcal{P}$.
 - **Case** $a = \frac{1}{2}$. Then $\varepsilon \cdot a = \frac{1}{2} = a$.
 - **Case** $a = \mathcal{P}$. Then $\varepsilon \cdot a = a$.
- **Condition** $a \leq b \Rightarrow a \cdot c \leq b \cdot c$. Pattern matching on a and b . If $a \leq b$, then either $b = \frac{1}{2}$ or there exists $\mathcal{P}, \mathcal{P}'$ such that $a = \mathcal{P}$ and $b = \mathcal{P}'$.
 - **Case** $b = \frac{1}{2}$. Then $b \cdot c = \frac{1}{2}$ is the top element, and then $a \cdot c \leq b \cdot c$.
 - **Case Otherwise**. $a \leq b$ iff $\mathcal{P} \preceq \mathcal{P}'$, then either $b \cdot c = \frac{1}{2}$ and $a \cdot c \leq b \cdot c$ follows, or $b \cdot c = \mathcal{P}' \otimes \mathcal{P}''$ for some probability space $c = \mathcal{P}''$. Then $\mathcal{P} \preceq \mathcal{P}'$ implies that $\mathcal{P} \cdot \mathcal{P}''$ is also defined and $\mathcal{P} \cdot \mathcal{P}' \preceq \mathcal{P} \cdot \mathcal{P}''$. Thus, $a \cdot c \leq b \cdot c$ too. \square

LEMMA D.2 (RA COMPOSITION PRESERVES COMPATIBILITY).

$$\mathcal{F}_1 \# p_1 \Rightarrow \mathcal{F}_2 \# p_2 \Rightarrow (\mathcal{F}_1 \oplus \mathcal{F}_2) \# (p_1 \cdot p_2)$$

PROOF. Let $S_1 = \{x \in \mathbb{X} \mid p_1(x) = 0\}$, $S_2 = \{x \in \mathbb{X} \mid p_2(x) = 0\}$. If $\mathcal{F}_1 \# p_1$, then there exists $\mathcal{P}'_1 \in \mathbb{P}(\mathbb{X} \setminus S_1 \rightarrow \mathbb{V})$ such that $\mathcal{P}_1 = \mathcal{P}'_1 \otimes \mathbb{1}_{S_1 \rightarrow \mathbb{V}}$. In addition, if $\mathcal{F}_2 \# p_2$, then there exists $\mathcal{P}'_2 \in \mathbb{P}(\mathbb{X} \setminus S_2 \rightarrow \mathbb{V})$ such that $\mathcal{P}_2 = \mathcal{P}'_2 \otimes \mathbb{1}_{S_2 \rightarrow \mathbb{V}}$. Then,

$$\begin{aligned} \mathcal{P}_1 \cdot \mathcal{P}_2 &= \mathcal{P}_1 \otimes \mathcal{P}_2 \\ &= (\mathcal{P}'_1 \otimes \mathbb{1}_{S_1 \rightarrow \mathbb{V}}) \otimes (\mathcal{P}'_2 \otimes \mathbb{1}_{S_2 \rightarrow \mathbb{V}}) \end{aligned}$$

Say $(\mathcal{F}'_1, \mu'_1) = \mathcal{P}'_1$, and $(\mathcal{F}'_2, \mu'_2) = \mathcal{P}'_2$. Then the sigma algebra of $\mathcal{P}_1 \cdot \mathcal{P}_2$ is

$$\begin{aligned} &\sigma(\{(E_1 \times S_1 \rightarrow \mathbb{V}) \cap (E_2 \times S_2 \rightarrow \mathbb{V}) \mid E_1 \in \mathcal{F}'_1, E_2 \in \mathcal{F}'_2\}) \\ &= \sigma(\{((E_1 \times (S_1 \setminus S_2) \rightarrow \mathbb{V}) \cap (E_2 \times (S_2 \setminus E_1) \rightarrow \mathbb{V})) \times (S_1 \cap S_2) \mid E_1 \in \mathcal{F}'_1, E_2 \in \mathcal{F}'_2\}) \end{aligned}$$

Then, there exists $\mathcal{P}'' \in \mathbb{P}(\mathbb{X} \setminus (S_1 \cap S_2) \rightarrow \mathbb{V})$ such that $\mathcal{P}_1 \cdot \mathcal{P}_2 = \mathcal{P}'' \otimes \mathbb{1}_{(S_1 \cap S_2) \rightarrow \mathbb{V}}$. Also,

$$\begin{aligned} &\{x \in \mathbb{X} \mid (p_1 \cdot p_2)(x) = 0\} \\ &= \{x \in \mathbb{X} \mid p_1(x) + p_2(x) = 0\} \\ &= \{x \in \mathbb{X} \mid p_1(x) = 0 \text{ and } p_2(x) = 0\} \\ &= S_1 \cap S_2 \end{aligned}$$

Therefore, $\mathcal{F}_1 \oplus \mathcal{F}_2$ is compatible with $p_1 \cdot p_2$ \square

LEMMA D.3. *The structure $(\text{Perm}, \preceq, \mathcal{V}, \cdot, \varepsilon)$ is an ordered unital resource algebra (RA) as defined in Definition 4.1.*

PROOF. We check the conditions one by one.

- **Condition** $a \cdot b = b \cdot a$. Follows from the commutativity of addition.
- **Condition** $(a \cdot b) \cdot c = a \cdot (b \cdot c)$. Follows from the associativity of addition.
- **Condition** $a \leq b \Rightarrow b \leq c \Rightarrow a \leq c$. \preceq is a point-wise lifting of the order \leq on arithmetics, so it follows from the transitivity of \leq .

- **Condition** $a \preceq a$. \preceq is a point-wise lifting of the order \leq on arithmetics, so it follows from the reflexivity of \leq .
- **Condition** $\mathcal{V}(a \cdot b) \Rightarrow \mathcal{V}(a)$. By definition,

$$\begin{aligned}
 \mathcal{V}(a \cdot b) &\Rightarrow \forall x \in \mathbb{X}, (a \cdot b)(x) \leq 1 \\
 &\Rightarrow \forall x \in \mathbb{X}, a(x) + b(x) \leq 1 \\
 &\Rightarrow \forall x \in \mathbb{X}, a(x) \leq 1 \\
 &\Rightarrow \mathcal{V}(a)
 \end{aligned}$$

- **Condition** $\mathcal{V}(\varepsilon)$. Note that $\varepsilon = \lambda x. 0$ satisfies that $\forall x \in \mathbb{X}, \varepsilon(x) \leq 1$, so $\mathcal{V}(\varepsilon)$.
- **Condition** $a \preceq b \Rightarrow \mathcal{V}(b) \Rightarrow \mathcal{V}(a)$. By definition, $a \preceq b$ means $\forall x \in \mathbb{X}, a(x) \leq b(x)$, and $\mathcal{V}(b)$ means that $\forall x \in \mathbb{X}, b(x) \leq 1$. Thus, $a \preceq b$ and $\mathcal{V}(b)$ implies that $\forall x \in \mathbb{X}, a(x) \leq b(x) \leq 1$, which implies $\mathcal{V}(a)$.
- **Condition** $\varepsilon \cdot a = a$. By definition,

$$\begin{aligned}
 \varepsilon \cdot a &= \lambda x. (\lambda x. 0)(x) + a(x) \\
 &= \lambda x. 0 + a(x) \\
 &= a.
 \end{aligned}$$

- **Condition** $a \preceq b \Rightarrow a \cdot c \preceq b \cdot c$. By definition,

$$\begin{aligned}
 a \preceq b &\Leftrightarrow \forall x \in \mathbb{X}, a(x) \leq b(x) \\
 &\Rightarrow \forall x \in \mathbb{X}, a(x) + c(x) \leq b(x) + c(x) \\
 &\Rightarrow a \cdot c \preceq b \cdot c
 \end{aligned}$$

□

LEMMA D.4. *The structure PSpPm is an ordered unital resource algebra (RA) as defined in Definition 4.1.*

PROOF. We want to check that PSpPm satisfies all the requirements to be an ordered unital resource algebra (RA). Because PSpPm is very close to a product of PSp and Perm, the proof below is very close to the proof that product RAs are RA.

First, Lemma D.2 implies that \cdot is well-defined.

Then we need to check all the RA axioms are satisfied. For any $a, b \in \text{PSpPm}$ and any $\mathcal{P}_1, p_1, \mathcal{P}_2, p_2$ such that $a = (\mathcal{P}_1, p_1), b = (\mathcal{P}_2, p_2)$.

We check the conditions one by one.

- **Condition** $\mathcal{V}(a \cdot b) \Rightarrow \mathcal{V}(a)$. By definition, $a \cdot b = (\mathcal{P}_1, p_1) \cdot (\mathcal{P}_2, p_2) = (\mathcal{P}_1 \cdot \mathcal{P}_2, p_1 \cdot p_2)$. And $\mathcal{V}(\mathcal{P}_1 \cdot \mathcal{P}_2, p_1 \cdot p_2)$ implies that $\mathcal{V}(\mathcal{P}_1 \cdot \mathcal{P}_2)$ and $\mathcal{V}(p_1 \cdot p_2)$. Because PSp and Perm are both RAs, we have $\mathcal{V}(\mathcal{P}_1)$ and $\mathcal{V}(p_1)$. Thus, $\mathcal{V}(\mathcal{P}_1, p_1)$.
- **Condition** $\mathcal{V}(\varepsilon)$. Clear because $\varepsilon = (\mathbb{1}_{\mathbb{S}}, \lambda x. 0)$ and $\mathbb{1}_{\mathbb{S}} \neq \perp$, and $\forall x. (\lambda x. 0)(x) \leq 1$.
- **Condition** $a \preceq b \Rightarrow \mathcal{V}(b) \Rightarrow \mathcal{V}(a)$. $a \preceq b$ implies that $\mathcal{P}_1 \preceq \mathcal{P}_2$ and $p_1 \preceq p_2$. $\mathcal{V}(b)$ implies that $\mathcal{P}_2 \neq \perp$, and $\forall x. (p_2)(x) \leq 1$. Thus, $\mathcal{P}_1 \neq \perp$, and $\forall x. (p_1)(x) \leq 1$. And therefore, $\mathcal{V}(b)$.
- **Condition** $\varepsilon \cdot a = a$. $\varepsilon \cdot a = (\mathbb{1}_{\mathbb{S}}, \lambda x. 0) \cdot (\mathcal{P}_1, p_1)$
 $= (\mathbb{1}_{\mathbb{S}} \cdot \mathcal{P}_1, \lambda x. 0 \cdot p_1)$
 $= (\mathcal{P}_1, p_1) = a$.
- **Condition** $a \preceq b \Rightarrow a \cdot c \preceq b \cdot c$. $a \preceq b$ implies that $\mathcal{P}_1 \preceq \mathcal{P}_2$ and $p_1 \preceq p_2$.
 Say $c = (\mathcal{P}_3, p_3)$. Then $a \cdot c = (\mathcal{P}_1 \cdot \mathcal{P}_3, p_1 \cdot p_3)$ and $b \cdot c = (\mathcal{P}_2 \cdot \mathcal{P}_3, p_2 \cdot p_3)$. Because $\mathcal{P}_1 \preceq \mathcal{P}_2$, $\mathcal{P}_1 \cdot \mathcal{P}_3 \preceq \mathcal{P}_2 \cdot \mathcal{P}_3$; similarly, $p_1 \cdot p_3 \preceq p_2 \cdot p_3$. Thus, $a \cdot c \preceq b \cdot c$. □

LEMMA D.5. *If M is an RA, then M^I is also an RA.*

PROOF. RA is known to be closed under products, and M^I can be obtained as products of M , so we omit the proof. \square

LEMMA D.6. \mathcal{M}_I is an RA.

PROOF. By Lemma D.4, PSpPm is an RA. By Lemma D.5, $\mathcal{M}_I = \text{PSpPm}^I$ is also an RA. \square

E Characterizations of Joint Conditioning and Relational Lifting

Interestingly, it is possible to characterize the conditioning modality using the other connectives of the logic.

PROPOSITION E.1 (ALTERNATIVE CHARACTERIZATION OF JOINT CONDITIONING). *The following is a logically equivalent characterization of the joint conditioning modality:*

$$\begin{aligned} C_\mu K \dashv \vdash \exists \mathcal{F}, \mu, p, \kappa. \text{Own}(\mathcal{F}, \mu, p) * \ulcorner \forall i \in I. \mu(i) = \text{bind}(\mu, \kappa(i)) \urcorner \\ * \forall v \in \text{supp}(\mu). \text{Own}(\mathcal{F}, \kappa(I)(v), p) \dashv \vdash K(v) \end{aligned}$$

PROOF. In the following, we sometimes abbreviate $\forall i \in I. \mu(i) = \text{bind}(\mu, \kappa(i))$ by writing just $\mu = \text{bind}(\mu, \kappa)$.

We start with the embedding:

$$\begin{aligned} & \exists \mathcal{F}, \mu, p, \kappa. \text{Own}(\mathcal{F}, \mu, p) * \ulcorner \forall i \in I. \mu(i) = \text{bind}(\mu, \kappa(i)) \urcorner \\ & \quad * \forall a \in \text{supp}(\mu). \text{Own}(\mathcal{F}, \kappa(I)(a), p) \dashv \vdash K(a) \\ \dashv \vdash & \lambda r. \exists \mathcal{F}, \mu, p, \kappa. (\text{Own}(\mathcal{F}, \mu', p) * \ulcorner \mu = \text{bind}(\mu, \kappa) \urcorner * \\ & \quad (\forall a \in \text{supp}(\mu). \text{Own}(\mathcal{F}, \kappa a, p) \dashv \vdash K(a)))(r) \\ \dashv \vdash & \lambda r. \exists \mathcal{F}, \mu, p, \kappa, \mathcal{F}_1, \mu_1, p_1, \mathcal{F}_2, \mu_2, p_2, \mathcal{F}_3, \mu_3, p_3, \\ & \quad r \sqsupseteq (\mathcal{F}_1, \mu_1, p_1) \cdot (\mathcal{F}_2, \mu_2, p_2) \cdot (\mathcal{F}_3, \mu_3, p_3) \wedge \\ & \quad (\mathcal{F}_1, \mu_1, p_1) \sqsupseteq (\mathcal{F}, \mu, p) \wedge \ulcorner \mu = \text{bind}(\mu, \kappa) \urcorner \wedge \\ & \quad (\forall a \in \text{supp}(\mu). \forall r_1, r_2. r_1 \cdot (\mathcal{F}_3, \mu_3, p_3) = r_2 \wedge r_1 \sqsupseteq (\mathcal{F}, \kappa a, p) \Rightarrow K(a)(r_2)) \\ \dashv \vdash & \lambda r. \exists \mathcal{F}, \mu, p, \mathcal{F}_3, \mu_3, p_3, \kappa. \\ & \quad r \sqsupseteq (\mathcal{F}, \mu, p) \cdot (\mathcal{F}_3, \mu_3, p_3) \wedge \ulcorner \mu = \text{bind}(\mu, \kappa) \urcorner \wedge \\ & \quad (\forall a \in \text{supp}(\mu). \forall r_1, r_2. r_1 \cdot (\mathcal{F}_3, \mu_3, p_3) = r_2 \wedge r_1 \sqsupseteq (\mathcal{F}, \kappa a, p) \Rightarrow K(a)(r_2)) \end{aligned}$$

For the last equivalence, the forward direction holds because

$$\begin{aligned} r & \sqsupseteq (\mathcal{F}_1, \mu_1, p_1) \cdot (\mathcal{F}_2, \mu_2, p_2) \cdot (\mathcal{F}_3, \mu_3, p_3) \\ & \sqsupseteq (\mathcal{F}_1, \mu_1, p_1) \cdot (\mathcal{F}_3, \mu_3, p_3) \\ & \sqsupseteq (\mathcal{F}, \mu, p) \cdot (\mathcal{F}_3, \mu_3, p_3). \end{aligned}$$

The backward direction holds because we can pick $(\mathcal{F}_1, \mu_1, p_1) = (\mathcal{F}, \mu, p)$, (\mathcal{F}_2, μ_2) be the trivial probability space on s and $p_2 = \lambda _ . 0$.

- To show that the embedding implies the original assertion $C_\mu K$, we start with $\mu(i) \otimes \mu_3(i)$. For any i , we have $\mu(i) = \text{bind}(\mu, \kappa(i))$, and thus

$$\mu(i) \otimes \mu_3(i) = \text{bind}(\mu, \kappa(i)) \otimes \mu_3(i).$$

According to Lemma C.7, $\mu(i) \otimes \mu_3(i)$ is defined implies that $\kappa(i)(a) \otimes \mu_3(i)$ is defined for any $a \in$. Furthermore,

$$\mu(i) \otimes \mu_3(i) = \text{bind}(\mu, \lambda a. \kappa(i)(a) \otimes \mu_3(i))$$

We abbreviate the hyperkernel $[i: \lambda a. \kappa(i)(a) \otimes \mu_3(i) \mid i \in I]$ as κ' . For any $a \in \text{supp}(\mu)$, the assertion

$$\forall a \in \text{supp}(\mu). \forall r_1, r_2. r_1 \otimes (\mathcal{F}_3, \mu_3, p_3) = r_2 \wedge r_1 \sqsupseteq (\mathcal{F}, \kappa(I)a, p) \Rightarrow K(a)(r_2)$$

applies with the specific case $r_1 = (\mathcal{F}, \kappa(I)(a), p)$, gives us

$$K(a)((\mathcal{F}, \kappa(I)(a), p) \cdot (\mathcal{F}_3, \mu_3, p_3))$$

By the definition of composition in our resource algebra, we have that $K(a)$ holds on $(\mathcal{F} \oplus \mathcal{F}_3, \kappa'(I)(a), p + p_3)$.

For any r ,

- If $\mathcal{V}(r)$, then there exists \mathcal{F}', μ', p' such that $r = (\mathcal{F}', \mu', p')$. Note that

$$r = (\mathcal{F}', \mu', p') \sqsupseteq (\mathcal{F}, \mu, p) \cdot (\mathcal{F}_3, \mu_3, p_3) = (\mathcal{F} \oplus \mathcal{F}_3, \mu \otimes \mu_3, p + p_3)$$

By Lemma C.4, $\mu \otimes \mu_3 = \text{bind}(\mu, \kappa')$ implies that there exists κ'' such that $\mu(i) = \text{bind}(\mu, \kappa''(i))$, and that for any $a \in \text{supp} \mu$, $(\mathcal{F} \oplus \mathcal{F}_3, \kappa'(I)(a)) \sqsubseteq (\mathcal{F}', \kappa''(I)(a))$. Thus, by monotonicity with respect to the extension order, that would imply $K(a)$ holds on $(\mathcal{F}', \kappa''(I)(a), p')$. And $K(a)$ holds on $(\mathcal{F}', \kappa''(I)(a), p')$ for any $a \in \text{supp} \mu$ together with $\mu(i) = \text{bind}(\mu, \kappa''(i))$ implies that r satisfy the original assertion of conditioning modality.

- If not $\mathcal{V}(r)$, then r satisfies any assertions, so r satisfy the original assertion of conditioning modality.
- To show the other direction that having the original assertion implies the embedded assertion. Assume $C_\mu K(r)$, that is,

$$\begin{aligned} \exists \mathcal{F}, \mu, p, \kappa. (\mathcal{F}, \mu, p) \preceq r \wedge \forall i \in I. \mu(i) = \text{bind}(\mu, \kappa(i))(r) \\ \wedge \forall v \in \text{supp}(\mu). K(v)(\mathcal{F}, \kappa(I)(v), p) \end{aligned}$$

To show that r also satisfy the embedding, we pick the witness for the existential quantifier as follows: let (\mathcal{F}_3, μ_3) be the trivial probability space on \mathbb{S} ; let $p_3 = \lambda_. 0$; pick $(\mathcal{F}_{\text{embd}}, \mu_{\text{embd}}, p_{\text{embd}})$ be the $(\mathcal{F}_{\text{orig}}, \mu_{\text{orig}}, p_{\text{orig}})$ that witness $C_\mu K(r)$, and $\kappa_{\text{embd}} = \kappa_{\text{orig}}$.

Then:

- First we show

$$\begin{aligned} r &\succeq (\mathcal{F}_{\text{orig}}, \mu_{\text{orig}}, p_{\text{orig}}) \\ &= (\mathcal{F}_{\text{orig}}, \mu_{\text{orig}}, p_{\text{orig}}) \cdot (\mathcal{F}_3, \mu_3, p_3) \\ &= (\mathcal{F}_{\text{embd}}, \mu_{\text{embd}}, p_{\text{embd}}) \cdot (\mathcal{F}_3, \mu_3, p_3) \end{aligned}$$

- $\mu_{\text{orig}} = \text{bind}(\mu, \kappa_{\text{orig}}(I)(a))$ implies $\mu_{\text{embd}} = \text{bind}(\mu, \kappa_{\text{embd}}(I)(a))$.
- For any r_1, r_2 ,

$$r_1 \cdot (\mathcal{F}_3, \mu_3, p_3) = r_2 \wedge r_1 \sqsupseteq (\mathcal{F}_{\text{embd}}, \kappa_{\text{embd}}(I)(a), p_{\text{embd}})$$

implies that $r_2 = r_1 \sqsupseteq (\mathcal{F}_{\text{orig}}, \kappa_{\text{orig}}(I)(a), p_{\text{orig}})$. By the assumption that the orig assertion holds, we have $K(a)(\mathcal{F}_{\text{orig}}, \kappa_{\text{orig}}(I)(a), p_{\text{orig}})$, which implies $K(a)(r_2)$.

Therefore, r also satisfy the embedding. \square

LEMMA E.2 (ALTERNATIVE CHARACTERIZATION OF RELATIONAL LIFTING). *Given a relation R over $\mathbb{S}_1 \times \mathbb{S}_2$, then $\llbracket R \rrbracket(\mathcal{F}, \mu, p)$ holds iff there exists $\hat{\mu}$ over $\mathcal{F}(1) \otimes \mathcal{F}(2)$ such that $\hat{\mu}(R) = 1$, $\hat{\mu} \circ \pi_1^{-1} = \mu(1)$, and $\hat{\mu} \circ \pi_2^{-1} = \mu(2)$.*

PROOF. We first unfold the definition of the coupling modality:

$$\begin{aligned}
& [R](\mathcal{F}, \mu, \mathbf{p}) \\
& \Leftrightarrow (\exists \mu. \mu(R) = 1 \wedge C_\mu \mathbf{v}. \bigwedge_{x\langle i \rangle \in X} \lceil x\langle i \rangle = v(x\langle i \rangle) \rceil)(\mathcal{F}, \mu, \mathbf{p}) \\
& \Leftrightarrow \exists \mu. \mu(R) = 1 \wedge (C_\mu \mathbf{v}. \bigwedge_{x\langle i \rangle \in X} \lceil x\langle i \rangle = v(x\langle i \rangle) \rceil)(\mathcal{F}, \mu, \mathbf{p}) \\
& \Leftrightarrow \exists \mu. \mu(R) = 1 \wedge \exists \mathcal{F}', \mu', \mathbf{p}', \kappa. (\mathcal{F}, \mu, \mathbf{p}) \succeq (\mathcal{F}', \mu', \mathbf{p}') \wedge \forall i \in I. \text{bind}(\mu, \kappa(i)) = \mu(i) \wedge \\
& \quad \forall v \in \text{supp}(\mu). (\bigwedge_{x\langle i \rangle \in X} \lceil x\langle i \rangle = v(x\langle i \rangle) \rceil)(\mathcal{F}', \mu', \mathbf{p}') \\
& \Leftrightarrow \exists \mu. \mu(R) = 1 \wedge \exists \mathcal{F}', \mu', \mathbf{p}', \kappa. (\mathcal{F}, \mu, \mathbf{p}) \succeq (\mathcal{F}', \mu', \mathbf{p}') \wedge \forall i \in I. \text{bind}(\mu, \kappa(i)) = \mu_i \wedge \\
& \quad \forall v \in \text{supp}(\mu). \bigwedge_{i \in \{1,2\}} \bigwedge_{x\langle i \rangle \in X} (x\langle i \rangle = v(x\langle i \rangle)) \prec (\mathcal{F}'(i), \kappa(i)(v))^\top \wedge \\
& \quad \lceil \kappa(i)(v) \circ (x\langle i \rangle = v(x\langle i \rangle))^{-1} = \delta_{\text{True}}^\top
\end{aligned}$$

Now, to show that $[R](\mathcal{F}, \mu, \mathbf{p})$ implies there exists $\widehat{\mu}$ over $\mathcal{F}(1) \otimes \mathcal{F}(2)$ such that $\widehat{\mu}(R) = 1$, $\widehat{\mu} \circ \pi_1^{-1} = \mu_1$, and $\widehat{\mu} \circ \pi_2^{-1} = \mu_2$, we define $\widehat{\mu}$ over $\mathcal{F}_1 \otimes \mathcal{F}_2$ as $\text{bind}(\mu, \lambda v. \kappa(1)(v) \otimes \kappa(2)(v))$. Then,

$$\begin{aligned}
\widehat{\mu}(R) &= \text{bind}(\mu, \lambda v. \kappa(1)(v) \otimes \kappa(2)(v))(R) \\
&= \sum_{v \in \text{supp}(\mu)} \mu(v) \cdot (\kappa(1)(v) \otimes \kappa(2)(v))(R)
\end{aligned}$$

Since $\mu(R) = 1$, then for all $v \in \text{supp}_\mu$, and $v \in R$, by additivity:

$$\begin{aligned}
(\kappa(1)(v) \otimes \kappa(2)(v))(R) &\geq (\kappa(1)(v) \otimes \kappa(2)(v))(v) \\
&= \kappa(1)(v)(\pi_1(v)) \cdot \kappa(2)(v)(\pi_2(v)) \\
&= \kappa(1)(v)(\bigwedge_{x\langle 1 \rangle \in X} x\langle 1 \rangle = v(x\langle 1 \rangle)) \cdot \kappa(2)(v)(\bigwedge_{x\langle 2 \rangle \in X} x\langle 2 \rangle = v(x\langle 2 \rangle)) \\
&= 1
\end{aligned} \tag{11}$$

where Eq. (11) is because v as a singleton can also be thought of as a Cartesian product. Thus,

$$\widehat{\mu}(R) = \sum_{v \in \text{supp}(\mu)} \mu(v) \cdot 1 \cdot 1 = \sum_{v \in \text{supp}(\mu)} \mu(v) = 1$$

Meanwhile, for $E_i \in \mathcal{F}_i$, and let $j = 2$ if $i = 1$ and $j = 1$ if $i = 2$,

$$\begin{aligned}
(\widehat{\mu} \circ \pi_i^{-1})(E_i) &= \widehat{\mu}(E_i \times \mathbb{S}_j) \\
&= \text{bind}(\mu, \lambda v. \kappa(1)(v) \otimes \kappa(2)(v))(E_i \times \mathbb{S}_j) \\
&= \sum_{v \in \text{supp } \mu} (\kappa(1)(v) \otimes \kappa(2)(v))(E_i \times \mathbb{S}_j) \\
&= \sum_{v \in \text{supp } \mu} \kappa(i)(v)(E_i) \cdot \kappa(j)(v)(\mathbb{S}_j) \\
&= \sum_{v \in \text{supp } \mu} \kappa(i)(v)(E_i) \cdot 1 \\
&= \text{bind}(\mu, \lambda v. \kappa(i))(E_i) \\
&= \mu_i(E_i)
\end{aligned} \tag{12}$$

Thus, we complete the forward direction.

For the backwards direction, we pick $\mu = \widehat{\mu}$, $\mu'(i) = \pi_i \widehat{\mu}(\mathcal{F}'(i))$ accordingly, $\mathbf{p}' = \mathbf{p}'$, and $\kappa(i) = \lambda v. \delta_{\pi_{x\langle i \rangle \in X} v}$. Then,

$$\begin{aligned}
\text{bind}(\mu, \kappa(i)) &= \text{bind}(\widehat{\mu}, \lambda v. \delta_{\pi_{x\langle i \rangle \in X} v}) \\
&= \widehat{\mu} \circ \pi_i^{-1} = \mu(i)
\end{aligned}$$

Also, by definition, $\mathbf{k}(i)(v) = \delta_{\pi_{\mathbf{x}(i)} \in X} v$. Thus, $\{\mathbf{x}\langle i \rangle = v(\mathbf{x}\langle i \rangle)\} \prec (\mathcal{F}(i), \kappa(i)(v))$ and $\kappa(i)(v) \circ \mathbf{x}\langle i \rangle = v(\mathbf{x}\langle i \rangle)^{-1} = \delta_{\text{True}}$. \square

F Soundness

F.1 Soundness of Primitive Rules

F.1.1 Soundness of Distribution Ownership Rules.

LEMMA F.1. *Section 4.2 is sound.*

PROOF. Assume a valid $a \in \mathcal{M}_I$ is such that it satisfies $P \wedge Q$. This means that for some $(\mathcal{F}, \mu, \mathbf{p}) \preceq a$, both $P(\mathcal{F}, \mu, \mathbf{p})$ and $Q(\mathcal{F}, \mu, \mathbf{p})$ hold. We want to prove $(P * Q)(a)$ holds. To this end, let $(\mathcal{F}_1, \mu_1, \mathbf{p}_1)$ and $(\mathcal{F}_2, \mu_2, \mathbf{p}_2)$ be such that for every $i \in \text{idx}(P)$:

$$\begin{array}{ll} \mathcal{F}_1(i) = \mathcal{F}(i) & \mathcal{F}_2(i) = \{\emptyset, \Omega\} \\ \mu_1(i) = \mu(i) & \mu_2(i) = \lambda X. \text{ if } X = \Omega \text{ then } 1 \text{ else } 0 \\ \mathbf{p}_1(i) = \mathbf{p}(i) & \mathbf{p}_2(i) = \lambda_{-}. 0 \end{array}$$

and for all $i \in I \setminus \text{idx}(P)$:

$$\begin{array}{ll} \mathcal{F}_2(i) = \mathcal{F}(i) & \mathcal{F}_1(i) = \{\emptyset, \Omega\} \\ \mu_2(i) = \mu(i) & \mu_1(i) = \lambda X. \text{ if } X = \Omega \text{ then } 1 \text{ else } 0 \\ \mathbf{p}_2(i) = \mathbf{p}(i) & \mathbf{p}_1(i) = \lambda_{-}. 0 \end{array}$$

Clearly, by construction, $(\mathcal{F}_1, \mu_1, \mathbf{p}_1) \otimes (\mathcal{F}_2, \mu_2, \mathbf{p}_2) = (\mathcal{F}, \mu, \mathbf{p})$, and $P(\mathcal{F}_1, \mu_1, \mathbf{p}_1)$. Since $\text{idx}(P) \cap \text{idx}(Q) = \emptyset$, we also have $Q(\mathcal{F}_2, \mu_2, \mathbf{p}_2)$. Therefore, $(P * Q)(\mathcal{F}, \mu, \mathbf{p})$, and so $(P * Q)(a)$ by upward closure. \square

LEMMA F.2. *Section A is sound.*

PROOF. Assume a valid $a \in \mathcal{M}_I$ is such that both $E\langle i \rangle \sim \mu(a)$ and $E\langle i \rangle \sim \mu'(a)$ hold. Let $a = (\mathcal{F}, \mu_0, \mathbf{p})$, then we know $\mu = \mu_0 \circ E\langle i \rangle^{-1} = \mu'$, which proves the claim. \square

LEMMA F.3. *Section A is sound.*

PROOF. The proof for the forward direction is very similar to the one for Section A. For $a \in \mathcal{M}_I$, if $(\lceil E_1\langle i \rangle \rceil * \lceil E_2\langle i \rangle \rceil)(a)$. Then there exists a_1, a_2 such that $a_1 \cdot a_2 \preceq a$ and $\lceil E_1\langle i \rangle \rceil(a_1), \lceil E_2\langle i \rangle \rceil(a_2)$. Say $a = (\mathcal{F}, \mu, \mathbf{p})$, $a_1 = (\mathcal{F}_1, \mu_1, \mathbf{p}_1)$ and $a_2 = (\mathcal{F}_2, \mu_2, \mathbf{p}_2)$. Then $\lceil E_1\langle i \rangle \rceil(a_1)$ implies that

$$\mu_1(E_1\langle i \rangle^{-1}(\text{True})) = 1$$

And similarly,

$$\mu_2(E_2\langle i \rangle^{-1}(\text{True})) = 1$$

Thus,

$$\mu(E_1\langle i \rangle^{-1}(\text{True}) \cap E_2\langle i \rangle^{-1}(\text{True})) = \mu_1(E_1\langle i \rangle^{-1}(\text{True})) \cdot \mu_2(E_2\langle i \rangle^{-1}(\text{True})) = 1.$$

Hence,

$$\mu(E_1\langle i \rangle \wedge E_2\langle i \rangle^{-1}(\text{True})) = \mu(E_1\langle i \rangle^{-1}(\text{True}) \cap E_2\langle i \rangle^{-1}(\text{True})) = 1$$

Thus, $\lceil E_1\langle i \rangle \wedge E_2\langle j \rangle \rceil(a)$.

Now we prove the backwards direction: Say $a = (\mathcal{F}, \mu, \mathbf{p})$. if $\lceil E_1\langle i \rangle \wedge E_2\langle j \rangle \rceil(a)$, then $\mu(E_1\langle i \rangle \wedge E_2\langle i \rangle^{-1}(\text{True})) = 1$, and then

$$\mu(E_1\langle i \rangle^{-1}(\text{True})) \geq \mu(E_1\langle i \rangle \wedge E_2\langle i \rangle^{-1}(\text{True})) = 1$$

$$\mu(E_2\langle i \rangle^{-1}(\text{True})) \geq \mu(E_1\langle i \rangle \wedge E_2\langle i \rangle^{-1}(\text{True})) = 1$$

Let $\mathcal{F}_1 = \sigma(E_1\langle i \rangle^{-1}(\text{True}))$ and $\mathcal{F}_2 = \sigma(E_2\langle i \rangle^{-1}(\text{True}))$. Then,

$$\begin{aligned} & \llbracket E_1\langle i \rangle \rrbracket(\mathcal{F}_1, \mu|_{\mathcal{F}_1}, \lambda_{-}, 0) \\ & \llbracket E_2\langle i \rangle \rrbracket(\mathcal{F}_2, \mu|_{\mathcal{F}_2}, \lambda_{-}, 0) \\ & (\mathcal{F}_1, \mu|_{\mathcal{F}_1}, \lambda_{-}, 0) * (\mathcal{F}_2, \mu|_{\mathcal{F}_2}, \lambda_{-}, 0) \preceq a \end{aligned}$$

Thus, $\llbracket E_1\langle i \rangle \rrbracket * \llbracket E_2\langle i \rangle \rrbracket$ holds on a . \square

LEMMA F.4. *Section A is sound.*

PROOF. Assume $a = (\mathcal{F}, \mu, \mathbf{p}) \in \mathcal{M}_I$ and $(\llbracket E\langle i \rangle \rrbracket \wedge P)(a)$ holds. We want to show that $(\llbracket E\langle i \rangle \rrbracket * P)(a)$ holds. First note that:

$$\begin{aligned} (\llbracket E\langle i \rangle \rrbracket \wedge P)(a) & \Rightarrow \llbracket E\langle i \rangle \rrbracket(a) \wedge P(a) \\ & \Rightarrow E \prec (\mathcal{F}(i), \mu(i)) \wedge \mu \circ E\langle i \rangle^{-1}(\text{true}) = \delta_{\text{True}} \wedge P(a) \end{aligned}$$

Define $\mathcal{F}', \mathbf{p}_E, \mathbf{p}_P$ such that, for any $j \in I$:

$$\begin{aligned} \mathcal{F}'(j) &= \begin{cases} \{\emptyset, \mathbb{S}\} & \text{if } j \neq i \\ \{\emptyset, \mathbb{S}, E\langle i \rangle^{-1}(\text{True}), \mathbb{S} \setminus E\langle i \rangle^{-1}(\text{True})\} & \text{otherwise} \end{cases} \\ \mathbf{p}_E(j) &= \begin{cases} \lambda_{-}.0 & \text{if } j \neq i \\ \lambda x\langle i \rangle. \text{if } x \in \text{pvar}(E) \text{ then } \mathbf{p}(i)(x\langle i \rangle)/2 \text{ else } 0 & \text{if } j = i \end{cases} \\ \mathbf{p}_P(j) &= \begin{cases} \mathbf{p}(j) & \text{if } j \neq i \\ \lambda x\langle i \rangle. \text{if } x \in \text{pvar}(E) \text{ then } \mathbf{p}(i)(x\langle i \rangle)/2 \text{ else } \mathbf{p}(i)(x\langle i \rangle) & \text{if } j = i \end{cases} \end{aligned}$$

By construction, we have $\mathbf{p} = \mathbf{p}_E \cdot \mathbf{p}_P$. Now let:

$$b = (\mathcal{F}', \mu|_{\mathcal{F}'}, \mathbf{p}_E) \quad a' = (\mathcal{F}, \mu, \mathbf{p}_P)$$

note that $\mathcal{V}(b)$ holds because $\mathcal{F}'(i)$ can at best be non-trivial on $\text{pvar}(E)$. The resource a' is also valid, since \mathbf{p}_P has the same non-zero components as \mathbf{p} . Then $\llbracket E\langle i \rangle \rrbracket(b)$ holds because $E \prec (\mathcal{F}'(i), \mu|_{\mathcal{F}'}(i))$ and $\mu|_{\mathcal{F}'} \circ E\langle i \rangle^{-1} = \mu \circ E\langle i \rangle^{-1} = \delta_{\text{True}}$. By applying Lemma C.6, it is easy to show that $(\mathcal{F}', \mu|_{\mathcal{F}'}) \otimes (\mathcal{F}, \mu)$ is defined and is equal to (\mathcal{F}, μ) . Therefore, $\mathcal{V}(b \cdot a)$ and $b \cdot a = a$. By the side condition $\text{pabs}(P, \text{pvar}(E\langle i \rangle))$ and the fact that \mathbf{p}_P is a scaled down version of \mathbf{p} , we obtain from $P(a)$ that $P(a')$ holds too. This proves that $(\llbracket E\langle i \rangle \rrbracket * P)(a)$ holds, as desired. \square

LEMMA F.5. *Section 5.2 is sound.*

PROOF. For any $(\mathcal{F}, \mu, \mathbf{p})$ such that $((E_1\langle i \rangle, E_2\langle i \rangle) \sim \mu_1 \otimes \mu_2)(\mathcal{F}, \mu, \mathbf{p})$, by definition, it must

$$\exists \mathcal{F}', \mu'. (\text{Own}(\mathcal{F}', \mu'))(\mathcal{F}, \mu, \mathbf{p}) * (E_1, E_2) \prec (\mathcal{F}'(i), \mu'(i)) \wedge \mu_1 \otimes \mu_2 = \mu'(i) \circ (E_1, E_2)^{-1}.$$

We can derive from it that

$$\begin{aligned} & \exists \mathcal{F}', \mu', \mathbf{p}'. (\mathcal{F}', \mu') \preceq (\mathcal{F}, \mu, \mathbf{p}) * \\ & \left(\forall a, b \in A. \exists L_{a,b}, U_{a,b} \in \mathcal{F}'(i). L_{a,b} \subseteq (E_1, E_2)^{-1}(a, b) \subseteq U_{a,b} \wedge \mu'(L_{a,b}) = \mu'(U_{a,b}) \wedge \right. \\ & \left. \mu_1 \otimes \mu_2(a, b) = \mu'(i)(L_{a,b}) = \mu'(i)(U_{a,b}) \right) \end{aligned}$$

Also, for any $a, b, a', b' \in A$ such that $a \neq a'$ or $b \neq b'$, we have $L_{a,b}$ disjoint from $L_{a',b'}$ because on $L_{a,b} \cap L_{a',b'}$, the random variable (E_1, E_2) maps to both (a, b) and (a', b') .

Define

$$\mathcal{F}_1(i) = \sigma(\{(\bigcup_{b \in A} L_{a,b}) \mid a \in A\} \cup \{(\bigcup_{b \in A} U_{a,b}) \mid a \in A\}),$$

and similarly define

$$\mathcal{F}_2(i) = \sigma(\{(\bigcup_{a \in A} L_{a,b}) \mid b \in A\} \cup \{(\bigcup_{a \in A} U_{a,b}) \mid b \in A\}).$$

Denote μ' restricted to \mathcal{F}_1 as μ'_1 and μ' restricted to \mathcal{F}_2 as μ'_2 .

We want to show that $(\mathcal{F}_1(i), \mu'_1(i)) \otimes (\mathcal{F}_2(i), \mu'_2(i)) \subseteq (\mathcal{F}'(i), \mu'(i))$, which boils down to show that for any $X_1 \in \mathcal{F}_1(i)$, any $X_2 \in \mathcal{F}_2(i)$,

$$\mu'(X_1 \cap X_2) = \mu'_1(X_1) \cdot \mu'_2(X_2)$$

For convenience, we will denote $\bigcup_{b \in A} L_{a,b}$ as L_a , denote $\bigcup_{a \in A} L_{a,b}$ as L_b , denote $\bigcup_{b \in A} U_{a,b}$ as U_a , and denote $\bigcup_{a \in A} U_{a,b}$ as U_b .

First, using a standard construction in measure theory proofs, we rewrite \mathcal{F}_1 and \mathcal{F}_2 as sigma algebra generated by sets of partitions. Specifically, \mathcal{F}_1 is equivalent to

$$\sigma(\{\bigcap_{a \in S_1} L_a \cap \bigcap_{a \in S_2} U_a \setminus (\bigcup_{a \in A \setminus S_1} L_a \cup \bigcup_{a \in A \setminus S_2} U_a) \mid S_1, S_2 \subseteq A\})$$

and similarly, \mathcal{F}_2 is equivalent to

$$\sigma(\{\bigcap_{b \in T_1} L_b \cap \bigcap_{b \in T_2} U_b \setminus (\bigcup_{b \in A \setminus T_1} L_b \cup \bigcup_{b \in A \setminus T_2} U_b) \mid T_1, T_2 \subseteq A\}).$$

Thus, by Lemma C.2, any event X_1 in \mathcal{F}_1 can be represented by

$$\biguplus_{S_1 \in I_1, S_2 \in I_2} \bigcap_{a \in S_1} L_a \cap \bigcap_{a \in S_2} U_a \setminus (\bigcup_{a \in A \setminus S_1} L_a \cup \bigcup_{a \in A \setminus S_2} U_a)$$

for some $I_1, I_2 \subseteq \mathcal{P}(A)$, where \mathcal{P} is the powerset over A . Similarly, any event X_2 in \mathcal{F}_2 can be represented by

$$\biguplus_{S_3 \in I_3, S_4 \in I_4} \bigcap_{b \in S_3} L_b \cap \bigcap_{b \in S_4} U_b \setminus (\bigcup_{b \in A \setminus S_3} L_b \cup \bigcup_{b \in A \setminus S_4} U_b)$$

for some $I_3, I_4 \subseteq \mathcal{P}(A)$. Thus, $X_1 \cap X_2$ can be represented as

$$\begin{aligned} X_1 \cap X_2 &= (\biguplus_{S_1 \in I_1, S_2 \in I_2} \bigcap_{a \in S_1} L_a \cap \bigcap_{a \in S_2} U_a \setminus (\bigcup_{a \in A \setminus S_1} L_a \cup \bigcup_{a \in A \setminus S_2} U_a)) \\ &\quad \cap (\biguplus_{S_3 \in I_3, S_4 \in I_4} \bigcap_{b \in S_3} L_b \cap \bigcap_{b \in S_4} U_b \setminus (\bigcup_{b \in A \setminus S_3} L_b \cup \bigcup_{b \in A \setminus S_4} U_b)) \\ &= \biguplus_{S_1 \in I_1, S_2 \in I_2, S_3 \in I_3, S_4 \in I_4} (\bigcap_{a \in S_1} L_a \cap \bigcap_{a \in S_2} U_a \setminus (\bigcup_{a \in A \setminus S_1} L_a \cup \bigcup_{a \in A \setminus S_2} U_a)) \\ &\quad \cap (\bigcap_{b \in S_3} L_b \cap \bigcap_{b \in S_4} U_b \setminus (\bigcup_{b \in A \setminus S_3} L_b \cup \bigcup_{b \in A \setminus S_4} U_b)) \end{aligned}$$

Because $L_{a,b}$ and $L_{a',b'}$ are disjoint as long as not $a = a'$ and $b = b'$, we have L_a disjoint from $L_{a'}$ if $a \neq a'$. Thus, $\bigcap_{a \in S_1} L_a \cap \bigcap_{a \in S_2} U_a \setminus (\bigcup_{a \in A \setminus S_1} L_a \cup \bigcup_{a \in A \setminus S_2} U_a)$ is not empty only when S_1 is singleton and empty.

- If S_1 is empty, then

$$\bigcap_{a \in S_1} L_a \cap \bigcap_{a \in S_2} U_a \setminus (\bigcup_{a \in A \setminus S_1} L_a \cup \bigcup_{a \in A \setminus S_2} U_a) = \bigcap_{a \in S_2} U_a \setminus (\bigcup_{a \in A} L_a \cup \bigcup_{a \in A \setminus S_2} U_a)$$

has measure 0 because $\bigcup_{a \in A} L_a$ has measure 1.

- Otherwise, if S_1 is singleton, say $S_1 = \{a'\}$, then

$$\bigcap_{a \in S_1} L_a \cap \bigcap_{a \in S_2} U_a \setminus (\bigcup_{a \in A \setminus S_1} L_a \cup \bigcup_{a \in A \setminus S_2} U_a) = L_{a'} \cap \bigcap_{a \in S_2} U_a \setminus (\bigcup_{a \in A \setminus S_2} U_a).$$

Furthermore,

$$\begin{aligned} \mu'(\bigcap_{a \in S_2} U_a) &= \mu'(\bigcap_{a \in S_2} L_a \uplus (U_a \setminus L_a)) \\ &= \mu'(\bigcap_{a \in S_2} L_a) + 0 \end{aligned}$$

And $\bigcap_{a \in S_2} L_a$ is non-empty only if S_2 is a singleton set or empty set. Thus, $L_{a'} \cap \bigcap_{a \in S_2} U_a \setminus (\bigcup_{a \in A \setminus S_2} U_a) \subseteq \bigcap_{a \in S_2} U_a$ has non-zero measure only if S_2 is empty or a singleton set.

- When S_2 is empty,

$$L_{a'} \cap \bigcap_{a \in S_2} U_a \setminus \bigcup_{a \in A \setminus S_2} U_a = L_{a'} \setminus \bigcup_{a \in A} U_a \subseteq L_{a'} \setminus U_{a'} = \emptyset$$

- When $S_2 = \{a'\}$,

$$L_{a'} \cap \bigcap_{a \in S_2} U_a \setminus \bigcup_{a \in A \setminus S_2} U_a = L_{a'} \setminus \bigcup_{a \in A, a \neq a'} U_a.$$

- When $S_2 = \{a''\}$ for some $a'' \neq a'$

$$\begin{aligned} L_{a'} \cap \bigcap_{a \in S_2} U_a \setminus \bigcup_{a \in A \setminus S_2} U_a &= L_{a'} \cap U_{a''} \setminus \bigcup_{a \in A, a \neq a''} U_a \\ &= \emptyset \end{aligned}$$

Thus,

$$\begin{aligned} \mu'(X_1) &= \mu' \left(\bigcup_{S_1 \in I_1, S_2 \in I_2} \bigcap_{a \in S_1} L_a \cap \bigcap_{a \in S_2} U_a \setminus (\bigcup_{a \in A \setminus S_1} L_a \cup \bigcup_{a \in A \setminus S_2} U_a) \right) \\ &= \mu' \left(\bigcup_{\{a'\} \in I_1, S_2 \in I_2} (L_{a'} \cap \bigcap_{a \in S_2} U_a \setminus \bigcup_{a \in A \setminus S_2} U_a) \right) \\ &= \mu' \left(\bigcup_{\{a'\} \in I_1 \cap I_2} L_{a'} \cap U_{a'} \setminus \bigcup_{a \in A, a \neq a'} U_a \right) \\ &= \mu' \left(\bigcup_{\{a'\} \in I_1 \cap I_2} (L_{a'} \setminus \bigcup_{a \in A, a \neq a'} U_a) \right) \\ &= \mu' \left(\bigcup_{\{a'\} \in I_1 \cap I_2} (L_{a'} \setminus \bigcup_{a \in A, a \neq a'} (L_a \cup (U_a \setminus L_a))) \right) \\ &= \mu' \left(\bigcup_{\{a'\} \in I_1 \cap I_2} (L_{a'} \setminus \bigcup_{a \in A, a \neq a'} (L_a)) \right) \\ &= \mu' \left(\bigcup_{\{a'\} \in I_1 \cap I_2} L_{a'} \right) \end{aligned}$$

Denote $\bigcup_{\{a'\} \in I_1 \cap I_2} L_{a'}$ as X'_1 . And $X_1 \setminus X'_1$ and $X'_1 \setminus X_1$ both have measure 0.

Similar results hold for X_2 as well, and we can show that

$$\mu'(X_2) = \mu' \left(\bigcup_{\{b'\} \in I_3 \cap I_4} L_{b'} \right)$$

Denote $\bigcup_{\{b'\} \in I_3 \cap I_4} L_{b'}$ as X'_2 . And $X_2 \setminus X'_2$ and $X'_2 \setminus X_2$ both have measure 0.

Thus,

$$\begin{aligned} \mu'(X_1 \cap X_2) &= \mu'(X_1 \cap X_2 \cap X'_1) + \mu'((X_1 \cap X_2) \setminus X'_1) \\ &= \mu'(X_1 \cap X_2 \cap X'_1) + 0 \\ &= \mu'(X_1 \cap X_2 \cap X'_1 \cap X'_2) + \mu'((X_1 \cap X_2 \cap X'_1) \setminus X'_2) + 0 \\ &= \mu'(X_1 \cap X_2 \cap X'_1 \cap X'_2) + 0 + 0 \\ &= \mu'(X_1 \cap X_2 \cap X'_1 \cap X'_2) + \mu'((X_2 \cap X'_1 \cap X'_2) \setminus X_1) \\ &= \mu'(X_2 \cap X'_1 \cap X'_2) \\ &= \mu'(X_2 \cap X'_1 \cap X'_2) + \mu'((X'_1 \cap X'_2) \setminus X_2) \\ &= \mu'(X'_1 \cap X'_2) \\ &= \mu'((\bigcup_{\{a'\} \in I_1 \cap I_2} L_{a'}) \cap (\bigcup_{\{b'\} \in I_3 \cap I_4} L_{b'})) \\ &= \mu'(\bigcup_{\{a'\} \in I_1 \cap I_2, \{b'\} \in I_3 \cap I_4} L_{a', b'}) \\ &= \sum_{\substack{\{a'\} \in I_1 \cap I_2 \\ \{b'\} \in I_3 \cap I_4}} \mu'(L_{a', b'}) \end{aligned}$$

Next we show that $\mu'(i)(L_{a,b}) = \mu'(i)(X_1) \cdot \mu'(i)(X_2)$. Note that $\mu'(L_a) = \sum_b \mu'(L_{a,b}) = \mu'(E_1^{-1}(a))$, and $\mu'(L_b) = \sum_a \mu'(L_{a,b}) = \mu'(E_2^{-1}(b))$. And $\mu_1 \otimes \mu_2 = \mu'(i) \circ (E_1, E_2)^{-1}$ implies that

$$\begin{aligned}\mu'(i)(L_{a,b}) &= \mu_1 \otimes \mu_2(a, b) \\ &= \mu_1(a) \cdot \mu_2(b)\end{aligned}$$

Then

$$\begin{aligned}\mu_1(a) &= \mu_1(a) \cdot \sum_{b \in A} \mu_2(b) \\ &= \sum_{b \in A} \mu_1(a) \cdot \mu_2(b) \\ &= \sum_{b \in A} \mu'(i)(L_{a,b}) \\ &= \mu'(i) \left(\sum_{b \in A} L_{a,b} \right) \\ &= \mu'(i)(L_a),\end{aligned}$$

and similarly,

$$\begin{aligned}\mu_2(b) &= \left(\sum_{a \in A} \mu_1(a) \right) \cdot \mu_2(b) \\ &= \sum_{a \in A} (\mu_1(a) \cdot \mu_2(b)) \\ &= \sum_{a \in A} \mu'(i)(L_{a,b}) \\ &= \mu'(i) \left(\sum_{a \in A} L_{a,b} \right) \\ &= \mu'(i)(L_b).\end{aligned}$$

Thus,

$$\mu'(i)(L_{a,b}) = \mu_1(a) \cdot \mu_2(b) = \mu'(i)(L_a) \cdot \mu'(i)(L_b)$$

Therefore,

$$\begin{aligned}\mu'(X_1 \cap X_2) &= \sum_{\substack{\{a'\} \in I_1 \cap I_2 \\ \{b'\} \in I_3 \cap I_4}} \mu'(L_{a',b'}) \\ &= \sum_{\substack{\{a'\} \in I_1 \cap I_2 \\ \{b'\} \in I_3 \cap I_4}} \mu'(L_{a'}) \cdot \mu'(L_{b'}) \\ &= \sum_{\{a'\} \in I_1 \cap I_2} \mu'(L_{a'}) \cdot \sum_{\{b'\} \in I_3 \cap I_4} \mu'(L_{b'}) \\ &= \mu'(X_1) \cdot \mu'(X_2) \\ &= \mu'_1(X_1) \cdot \mu'_2(X_2)\end{aligned}$$

Thus we have $(\mathcal{F}_1, \mu'_1) \otimes (\mathcal{F}_2, \mu'_2) \sqsubseteq (\mathcal{F}', \mu')$. Let $p_1 = p_2 = \lambda x. p'(x)/2$.

Next we show that $E_1 \sim \mu_1(\mathcal{F}_1, \mu'_1, \mathbf{p}_1)$ and $E_2 \sim \mu_2(\mathcal{F}_2, \mu'_2, \mathbf{p}_2)$. By definition, $E_1 \sim \mu_1(\mathcal{F}_1, \mu'_1, \mathbf{p}_1)$ is equivalent to

$$\exists \mathcal{F}'', \mu''. (\text{Own}(\mathcal{F}'', \mu''))(\mathcal{F}_1, \mu'_1, \mathbf{p}_1) * E_1 \prec (\mathcal{F}''(i), \mu''(i)) \wedge \mu_1 = \mu''(i) \circ E_1^{-1},$$

which is equivalent to

$$\begin{aligned} \exists \mathcal{F}'', \mu''. (\mathcal{F}'', \mu'') \preceq (\mathcal{F}_1, \mu'_1) * (\forall a \in A. \exists S_a, T_a \in \mathcal{F}''(i). \\ S_a \subseteq E_1^{-1}(a) \subseteq T_a \wedge \mu''(i)(S_a) = \mu''(i)(T_a) \wedge \mu_1(a) = \mu''(i)(S_a) = \mu''(i)(T_a)) \end{aligned}$$

We can pick the existential witness to be \mathcal{F}_1, μ'_1 . For any $a \in A$, $E_1^{-1}(a) = \bigcup_{b \in A} (E_1, E_2)^{-1}(a, b)$. Because we have $L_{a,b} \subseteq (E_1, E_2)^{-1}(a, b) \subseteq U_{a,b}$, then

$$\bigcup_{b \in A} L_{a,b} \subseteq E_1^{-1}(a) = \bigcup_{b \in A} (E_1, E_2)^{-1}(a, b) \subseteq \bigcup_{b \in A} U_{a,b}.$$

By definition, for each a , $\bigcup_{b \in A} L_{a,b} \in \mathcal{F}_1(i)$ and $\bigcup_{b \in A} U_{a,b} \in \mathcal{F}_1(i)$, and we also have

$$\begin{aligned} \mu'_1(i)(\bigcup_{b \in A} L_{a,b}) &= \sum_{b \in A} \mu'_1(i)(L_{a,b}) \\ &= \sum_{b \in A} \mu'_1(i)(U_{a,b}) \\ &= \mu'_1(i)(\bigcup_{b \in A} U_{a,b}) \\ &= \mu_1(a) \end{aligned}$$

Thus, $S_a = \bigcup_{b \in A} L_{a,b}$ and $T_a = \bigcup_{b \in A} U_{a,b}$ witnesses the conditions needed for $E_1 \sim \mu_1(\mathcal{F}_1, \mu'_1, \mathbf{p}_1)$. And similarly, we have $E_2 \sim \mu_2(\mathcal{F}_2, \mu'_2, \mathbf{p}_2)$. \square

F.1.2 Soundness of Conditioning Rules.

LEMMA F.6. *Section 4.3 is sound.*

PROOF. Let $\varepsilon = (\mathcal{F}_\varepsilon, \mu_\varepsilon, \mathbf{p}_\varepsilon) \in \mathcal{M}_I$ be the unit of \mathcal{M}_I and $\kappa = \lambda v. \mu_\varepsilon$. Then,

$$\begin{aligned} \text{True} &\vdash \text{Own}(\mathcal{F}_\varepsilon, \mu_\varepsilon) \\ &\vdash \text{Own}(\mathcal{F}_\varepsilon, \mu_\varepsilon) * \ulcorner \forall i \in I. \mu_\varepsilon(i) = \text{bind}(\mu, \kappa(i)) \urcorner \\ &\vdash \text{Own}(\mathcal{F}_\varepsilon, \mu_\varepsilon) * \ulcorner \forall i \in I. \mu_\varepsilon(i) = \text{bind}(\mu, \kappa(i)) \urcorner * \text{True} \\ &\vdash \exists \mathcal{F}_\varepsilon, \mu_\varepsilon, \kappa. \text{Own}(\mathcal{F}_\varepsilon, \mu_\varepsilon) * \ulcorner \forall i \in I. \mu_\varepsilon(i) = \text{bind}(\mu, \kappa(i)) \urcorner \\ &\quad * (\forall v \in \text{supp}(\mu). \text{Own}(\mathcal{F}_\varepsilon, \kappa(I)(v), \mathbf{p}_\varepsilon) -* \text{True}) \\ &\vdash C_{\mu _}. \text{True} \end{aligned} \quad \square$$

LEMMA F.7. *Section A is sound.*

PROOF. Assume $a \in \mathcal{M}_I$ is such that $\mathcal{V}(a)$ and that it satisfies $C_\mu v$. False. By definition, this means that, for some $\mathcal{F}_0, \mu_0, \mathbf{p}_0$, and κ_0 :

$$(\mathcal{F}_0, \mu_0, \mathbf{p}_0) \preceq a \quad (13)$$

$$\forall i \in I. \mu_0(i) = \text{bind}(\mu, \kappa_0(i)) \quad (14)$$

$$\forall v \in \text{supp}(\mu). \text{False}(\mathcal{F}_0, \kappa_0(I)(v), \mathbf{p}_0) \quad (15)$$

Let $v_0 \in \text{supp}(\mu)$ —we know one exists because μ is a (discrete) probability distribution. Then by (15) on v_0 we get $\text{False}(\mathcal{F}_0, \kappa_0(I)(v_0), \mathbf{p}_0)$ holds. Since $\text{False}(_)$ is by definition false, we get $\text{False}(a)$ holds *ex falso*. \square

LEMMA F.8. *Section 2.3 is sound.*

PROOF. Assume $a \in \mathcal{M}_I$ is such that $\mathcal{V}(a)$ and that it satisfies $C_\mu v. K(v)$. By definition, this means that, for some $\mathcal{F}_0, \mu_0, p_0$, and κ_0 :

$$(\mathcal{F}_0, \mu_0, p_0) \preceq a \quad (16)$$

$$\forall i \in I. \mu_0(i) = \text{bind}(\mu, \kappa_0(i)) \quad (17)$$

$$\forall v \in \text{supp}(\mu). K(v)(\mathcal{F}_0, \kappa_0(I)(v), p_0) \quad (18)$$

Then by the premise $\forall v. K(v) \vdash K'(v)$ and (18) we obtain

$$\forall v \in \text{supp}(\mu). K'(v)(\mathcal{F}_0, \kappa_0(I)(v), p_0) \quad (19)$$

By (16), (17), and (19) we get $C_\mu v. K'(v)$ as desired. \square

LEMMA F.9. Section 2.3 is sound.

PROOF. Assume $a \in \mathcal{M}_I$ is such that $\mathcal{V}(a)$ and that it satisfies $P * C_\mu v. K(v)$. By definition, this means that there exist some $(\mathcal{F}_1, \mu_1, p_1)$, $(\mathcal{F}_2, \mu_2, p_2)$, and κ such that

$$(\mathcal{F}_1, \mu_1, p_1) \cdot (\mathcal{F}_2, \mu_2, p_2) \preceq a \quad (20)$$

$$P(\mathcal{F}_1, \mu_1, p_1) \quad (21)$$

$$\forall i \in I. \mu_2(i) = \text{bind}(\mu, \kappa(i)) \quad (22)$$

$$\forall v \in \text{supp}(\mu). K(v)(\mathcal{F}_2, \kappa(I)(v), p_2) \quad (23)$$

Now let:

$$(\mathcal{F}', \mu', p') = (\mathcal{F}_1(i), \mu_1(i)) \otimes (\mathcal{F}_2(i), \mu_2(i)) \quad \kappa'(i) = \lambda v. \mu_1(i) \otimes \kappa(i)(v)$$

By Lemma C.7, for each $i \in I$:

$$\begin{aligned} (\mathcal{F}', \mu', p') &= (\mathcal{F}_1(i), \mu_1(i)) \otimes (\mathcal{F}_2(i), \mu_2(i)) \\ &= (\mathcal{F}_1(i) \oplus \mathcal{F}_2(i), \text{bind}(\mu, \lambda v. \mu_1(i) \otimes \kappa(i)(v))) \quad (\text{By Lemma C.7}) \\ &= (\mathcal{F}_1(i) \oplus \mathcal{F}_2(i), \text{bind}(\mu, \kappa'(i))) \end{aligned}$$

Notice that $\kappa'(I)(v) = \mu_1 \otimes \kappa(I)(v)$. Thus we obtain:

$$(\mathcal{F}', \mu', p') \preceq a \quad (24)$$

$$\forall i \in I. \mu'(i) = \text{bind}(\mu, \kappa'(i)) \quad (25)$$

and for all $v \in \text{supp}(\mu)$,

$$(\mathcal{F}_1, \mu_1, p_1) \otimes (\mathcal{F}_2, \kappa(I)(v), p_2) = (\mathcal{F}', \mu_1 \otimes \kappa(I)(v), p') \preceq (\mathcal{F}', \kappa'(I)(v), p') \quad (26)$$

$$P(\mathcal{F}_1, \mu_1, p_1) \quad (27)$$

$$K(v)(\mathcal{F}_2, \kappa(I)(v), p_2) \quad (28)$$

which gives us that a satisfies $C_\mu v. (P * K(v))$ as desired. \square

LEMMA F.10. Section 4.3 is sound.

PROOF. Straightforward. \square

LEMMA F.11. Section 2.3 is sound.

PROOF. We prove the two directions separately.

- **Forward direction** $E\langle i \rangle \sim \mu \vdash C_\mu v. \lceil E\langle i \rangle = v \rceil$. By unfolding the assumption $E\langle i \rangle \sim \mu$ we get that there exist \mathcal{F}, μ such that:

$$\text{Own}(\mathcal{F}, \mu) * \lceil E \prec (\mathcal{F}(i), \mu(i)) \rceil * \lceil \mu = \mu(i) \circ E^{-1} \rceil$$

holds. Let

$$\kappa \triangleq \lambda j. \begin{cases} \lambda v. \mu(j) & \text{if } j \neq i \\ \lambda v. \gamma_v & \text{if } j = i \end{cases} \quad \gamma_v \triangleq \lambda X : \mathcal{F}(i). \frac{\mu(i)(X \cap (E = v)^{-1})}{\mu(i)((E = v)^{-1})}$$

That is, $\kappa(j)$ maps every v to $\mu(j)$ when $i \neq j$, while when $i = j$ it maps v to the distribution $\mu(i)$ conditioned on $E = v$. Note that κ is well defined because (1) although the events $X \cap (E = v)^{-1}$ and $(E = v)^{-1}$ might not belong to $\mathcal{F}(i)$, their probability is uniquely determined by almost measurability of E ; (2) we are only interested in the cases where $v \in \text{supp}(\mu)$, which implies that the denominator is not zero: $\mu(i)((E = v)^{-1}) = \mu(v) > 0$. By construction we obtain that

$$\forall j \in I. \mu(j) = \text{bind}(\mu, \kappa(j)) \quad (29)$$

$$\forall v \in \text{supp}(\mu). \kappa(i)(v)((E = v)^{-1}) = 1 \quad (30)$$

From (30) we get that $\lceil E\langle i \rangle = v \rceil$ holds on $(\mathcal{F}(i), \kappa(i)(v), p(i))$, from which it follows that:

$$\text{Own}(\mathcal{F}, \kappa(I)(v), p) \multimap \lceil E\langle i \rangle = v \rceil$$

Therefore we obtain

$$\begin{aligned} \exists \mathcal{F}, \mu, \kappa, p. \text{Own}(\mathcal{F}, \mu, p) * \lceil \forall j \in I. \mu(j) = \text{bind}(\mu, \kappa(j)) \rceil \\ * (\forall v \in A_\mu. \text{Own}(\mathcal{F}, \kappa(I)(v), p) \multimap \lceil E\langle i \rangle = v \rceil) \end{aligned}$$

which gives us $C_\mu v. \lceil E\langle i \rangle = v \rceil$ by Proposition E.1.

- **Backward direction** $C_\mu v. \lceil E\langle i \rangle = v \rceil \vdash E\langle i \rangle \sim \mu$. First note that

$$\begin{aligned} \lceil E\langle i \rangle = v \rceil(\mathcal{F}, \kappa(v), p) \\ \Leftrightarrow ((E = v) \in \text{true})\langle i \rangle \sim \delta_{\text{true}}(\mathcal{F}, \kappa(I)(v), p) \\ \Leftrightarrow ((E = v) \in \text{true}) \prec (\mathcal{F}(i), \kappa(i)(v)) \wedge \delta_{\text{true}} = \kappa(i)(v) \circ ((E = v) \in \text{true})^{-1} \\ \Leftrightarrow ((E = v) \in \text{true}) \prec (\mathcal{F}(i), \kappa(i)(v)) \wedge \delta_v = \kappa(i)(v) \circ E^{-1} \end{aligned}$$

for some κ . This implies $\lceil E \prec \mathcal{F}(i), \kappa(i)(v) \rceil$. Then, for any value $v \in \text{supp}(\mu)$,

$$\begin{aligned} \mu(i) \circ E^{-1}(v) &= (\text{bind}(\mu, \kappa(i)) \circ E^{-1})(v) \\ &= \text{bind}(\mu, \kappa(i))(E^{-1}(v)) \\ &= \sum_{v' \in \text{supp}(\mu)} \mu(v') \cdot \kappa(i)(v')(E^{-1}(v)) \\ &= \sum_{v' \in \text{supp}(\mu)} \mu(v') \cdot (\kappa(i)(v') \circ E^{-1})(v) \\ &= \sum_{v' \in \text{supp}(\mu)} \mu(v') \cdot \delta_{v'}(v) \\ &= \mu(v) \end{aligned}$$

This implies the pure facts that $E \prec (\mathcal{F}(i), \mu(i))$ and $\mu = \mu(i) \circ E^{-1}$. Therefore:

$$\begin{aligned} C_\mu v. \lceil E\langle i \rangle = v \rceil \vdash \exists \mathcal{F}, \mu, \kappa, p. \text{Own}(\mathcal{F}, \mu, p) * \lceil \forall j \in I. \mu(j) = \text{bind}(\mu, \kappa(j)) \rceil \\ * (\forall v \in A_\mu. \text{Own}(\mathcal{F}, \kappa(I)(v), p) \multimap \lceil E\langle i \rangle = v \rceil) \end{aligned}$$

$$\begin{aligned} & \vdash \exists \mathcal{F}, \mu. \text{Own}(\mathcal{F}, \mu) * \lceil E \prec (\mathcal{F}(i), \mu(i)) \rceil * \lceil \mu = \mu(i) \circ E^{-1} \rceil \\ & \vdash E\langle i \rangle \sim \mu \end{aligned} \quad \square$$

LEMMA F.12. *Section A is sound.*

PROOF. Define $\kappa' = \lambda v. \text{bind}(\kappa(v), \lambda w. \text{return}(v, w))$. We start by rewriting the assumption $C_\mu v. C_{\kappa(v)} w. K(v, w)$ so that k' is used and K depends only on the binding of the innermost modality:

$$\begin{aligned} C_\mu v. C_{\kappa(v)} w. K(v, w) & \vdash C_\mu v. C_{\kappa'(v)}(v', w). K(v, w) & (\text{C-TRANSF, C-CONS}) \\ & \vdash C_\mu v. C_{\kappa'(v)}(v', w). K(v', w) & (\text{C-PURE, C-CONS}) \end{aligned}$$

Section 4.3 is applied to the innermost modality by using the bijection $f_v(w) = (v, w)$. Then, since $(v', w) \in \text{supp}(k'(v)) \Rightarrow v = v'$, we can replace v' for v in K .

Our goal is now to prove:

$$C_\mu v. C_{\kappa'(v)}(v', w). K(v', w) \vdash C_{\text{bind}(\mu, \kappa')}(v', w). K(v', w)$$

Let $a \in \mathcal{M}_I$ be such that $\mathcal{V}(a)$ and that it satisfies $C_\mu v. C_{\kappa'(v)}(v', w). K(v', w)$. From this assumption we know that, for some $\mathcal{F}_0, \mu_0, p_0$, and κ_0 :

$$(\mathcal{F}_0, \mu_0, p_0) \preceq a \quad (31)$$

$$\forall i \in I. \mu_0(i) = \text{bind}(\mu, \kappa_0(i)) \quad (32)$$

such that $\forall v \in \text{supp}(\mu)$, there are some $\mathcal{F}_1^v, \mu_1^v, p_1^v$, and κ_1^v satisfying:

$$(\mathcal{F}_1^v, \mu_1^v, p_1^v) \preceq (\mathcal{F}_0, \kappa_0(I)(v), p_0) \quad (33)$$

$$\forall i \in I. \mu_1^v(i) = \text{bind}(\kappa'(v), \kappa_1^v(i)) \quad (34)$$

$$\forall (v', w) \in \text{supp}(\kappa'(v)). K(v', w) (\mathcal{F}_1^v, \kappa_1^v(I)(v', w), p_1^v) \quad (35)$$

Our goal is to prove $C_{\text{bind}(\mu, \kappa')}(v', w). K(v', w)$ holds on a . To this end, we want to show that there exists κ'_2 such that:

$$\forall i \in I. \mu_0(i) = \text{bind}(\text{bind}(\mu, \kappa'), \kappa'_2(i)) \quad (36)$$

$$\forall (v', w) \in \text{supp}(\text{bind}(\mu, \kappa')). K(v', w) (\mathcal{F}_0, \kappa'_2(I)(v'), p_0) \quad (37)$$

Now let

$$\kappa_2(i) = \lambda(v', w). \kappa_1^{v'}(i)(v', w).$$

which by construction and Eq. (34) gives us

$$\mu_1^v(i) = \text{bind}(\kappa'(v), \kappa_1^v(i)) = \text{bind}(\kappa'(v), \kappa_2(i))$$

Therefore, by Eq. (33), we can apply Lemma C.4 and obtain that there exists a κ'_2 such that

$$\kappa_0(i)(v) = \text{bind}(\kappa'(v), \kappa'_2(i)) \quad (38)$$

$$(\mathcal{F}_0, \kappa'_2(i)(v', w)) \supseteq (\mathcal{F}_1^{v'}, \kappa_2(i)(v', w)) = (\mathcal{F}_1^{v'}, \kappa_1^{v'}(i)(v', w)) \quad (39)$$

By Eqs. (32) and (38) we have:

$$\begin{aligned} \mu_0(i) &= \text{bind}(\mu, \kappa_0(i)) \\ &= \text{bind}(\mu, \lambda v. \text{bind}(\kappa'(v), \kappa'_2(i))) & \text{By (ASSOC)} \\ &= \text{bind}(\text{bind}(\mu, \kappa'), \kappa'_2(i)) \end{aligned}$$

which proves Eq. (36).

Finally, to prove Eq. (37), we can observe that $(v', w) \in \text{supp}(\text{bind}(\mu, \kappa'))$ implies $v' \in \text{supp}(\mu)$; therefore, by (35), upward closure of $K(v', w)$, and (39) and (33), we can conclude $K(v', w)$ holds on $(\mathcal{F}_0, \kappa'_2(I)(v'), p_0)$, as desired. \square

LEMMA F.13. *Section 5.3 is sound.*

PROOF. Assume $a \in \mathcal{M}_I$ is such that $\mathcal{V}(a)$ and that it satisfies $\mathcal{C}_{\text{bind}(\mu, \kappa)} w. K(w)$. By definition, this means that, for some $\mathcal{F}_0, \mu_0, p_0$, and κ_0 :

$$(\mathcal{F}_0, \mu_0, p_0) \preceq a \quad (40)$$

$$\forall i \in I. \mu_0(i) = \text{bind}(\text{bind}(\mu, \kappa), \kappa_0(i)) \quad (41)$$

$$\forall w \in \text{supp}(\text{bind}(\mu, \kappa)). K(w)(\mathcal{F}_0, \kappa_0(I)(w), p_0) \quad (42)$$

Our goal is to show that a satisfies $\mathcal{C}_\mu v. \mathcal{C}_{\kappa(v)} w. K(w)$, for which it would suffice to show that there is a κ_1 such that:

$$\forall i \in I. \mu_0(i) = \text{bind}(\mu, \kappa_1(i)) \quad (43)$$

and for all $v \in \text{supp}(\mu)$ there is a κ_2^v with

$$\forall i \in I. \kappa_1(i)(v) = \text{bind}(\kappa(v), \kappa_2^v(i)) \quad (44)$$

$$\forall w \in \text{supp}(\kappa(v)). K(w)(\mathcal{F}_0, \kappa_2^v(I)(w), p_0) \quad (45)$$

To prove this we let

$$\kappa_1(i) = \lambda v. \text{bind}(\kappa(v), \kappa_0(i)) \quad \kappa_2^v(i) = \kappa_0(i)$$

By (ASSOC) we have

$$\mu_0(i) = \text{bind}(\text{bind}(\mu, \kappa), \kappa_0(i)) = \text{bind}(\mu, \lambda v. \text{bind}(\kappa(v), \kappa_0(i))) = \text{bind}(\mu, \kappa_1(i))$$

which proves (43). By construction,

$$\kappa_1(i)(v) = \text{bind}(\kappa(v), \kappa_0(i)) = \text{bind}(\kappa(v), \kappa_2^v(i))$$

proving (44). Finally, $v \in \text{supp}(\mu)$ and $w \in \text{supp}(\kappa(v))$ imply $w \in \text{supp}(\text{bind}(\mu, \kappa))$, so by (42) we proved (45), concluding the proof. \square

LEMMA F.14. *Section 4.3 is sound.*

PROOF. Let $I_1 = \text{idx}(K_1)$ and $I_2 = I \setminus I_1$; by $\text{idx}(K_1) \cap \text{idx}(K_2) = \emptyset$ we have $I_2 \supseteq \text{idx}(K_2)$. Assume $a \in \mathcal{M}_I$ is such that $\mathcal{V}(a)$ holds and that it satisfies $\mathcal{C}_\mu v. K_1(v) \wedge \mathcal{C}_\mu v. K_2(v)$. This means that for each $j \in \{1, 2\}$, for some $\mathcal{F}_j, \mu_j, p_j$, and κ_j :

$$(\mathcal{F}_j, \mu_j, p_j) \preceq a \quad (46)$$

$$\forall i \in I. \mu_j(i) = \text{bind}(\mu, \kappa_j(i)) \quad (47)$$

$$\forall v \in \text{supp}(\mu). K_j(v)(\mathcal{F}_j, \kappa_j(I)(v), p_j) \quad (48)$$

Now let

$$\hat{\mathcal{F}} = \begin{cases} \mathcal{F}_1(i) & \text{if } i \in I_1 \\ \mathcal{F}_2(i) & \text{if } i \in I_2 \end{cases} \quad \hat{\mu} = \begin{cases} \mu_1(i) & \text{if } i \in I_1 \\ \mu_2(i) & \text{if } i \in I_2 \end{cases} \quad \hat{p} = \begin{cases} p_1(i) & \text{if } i \in I_1 \\ p_2(i) & \text{if } i \in I_2 \end{cases} \quad \hat{\kappa}(i) = \begin{cases} \kappa_1(i) & \text{if } i \in I_1 \\ \kappa_2(i) & \text{if } i \in I_2 \end{cases}$$

By construction, we have:

$$(\hat{\mathcal{F}}, \hat{\mu}, \hat{p}) \preceq a$$

$$\forall i \in I. \hat{\mu}(i) = \text{bind}(\mu, \hat{\kappa}(i))$$

Moreover, for any $v \in \text{supp}(\mu)$ and any $j \in \{1, 2\}$, since $I_j \supseteq \text{idx}(K_j)$, condition (48) implies

$$K_j(v)(\hat{\mathcal{F}}, \hat{\kappa}(I)(v), \hat{p})$$

This means $(\hat{\mathcal{F}}, \hat{\kappa}(I)(v), \hat{p})$ satisfies $(K_1(v) \wedge K_2(v))$, and thus a satisfies $C_\mu v. (K_1(v) \wedge K_2(v))$, as desired. \square

LEMMA F.15. *Section 2.3 is sound.*

PROOF. For any resource $r = (\mathcal{F}, \mu, p)$,

$$\begin{aligned} & (C_\mu v. \exists x : \mathbb{X}. Q(v, x))(\mathcal{F}, \mu, p) \\ & \Leftrightarrow \exists \kappa. \forall i \in I. \mu(i) = \text{bind}(\mu, \kappa(i)) \wedge \forall v \in \text{supp}(\mu). (\exists x : \mathbb{X}. Q(v, x))(\mathcal{F}, \kappa(I)(v), p) \end{aligned}$$

For all $v \in \text{supp}(\mu)$, $\exists x : \mathbb{X}. Q(v, x)$ holds on $(\mathcal{F}, \kappa(I)(v), p)$. Thus, $Q(v, x_v)(\mathcal{F}, \kappa(I)(v), p)$ holds for some x_v . Then define $f : A \rightarrow \mathbb{X}$ by letting $f(v) = x_v$ for $v \in \text{supp}(\mu)$. Then,

$$\exists \kappa. \forall i \in I. \mu(i) = \text{bind}(\mu, \kappa(i)) \wedge \forall v \in \text{supp}(\mu). Q(v, f(v))(\mathcal{F}, \kappa(I)(v), p)$$

And therefore \mathcal{F}, μ, p satisfies $\exists f : A \rightarrow \mathbb{X}. C_\mu v. Q(v, x)$. \square

LEMMA F.16. *Section 4.3 is sound.*

PROOF. For any resource $a = (\mathcal{F}, \mu, p)$, if $(C_\mu v. K(v))((\mathcal{F}, \mu, p))$, then

$$\begin{aligned} & \exists \kappa. (\mathcal{F}, \mu, p) \preceq a \wedge \forall i \in I. \mu(i) = \text{bind}(\mu, \kappa(i)) \\ & \wedge \forall v \in \text{supp}(\mu). (K(v))((\mathcal{F}, \kappa(I)(v), p)) \end{aligned}$$

$\mu = \text{bind}(\mu, \kappa)$ says that for any $E \in \mathcal{F}$,

$$\begin{aligned} \mu(E) &= \sum_{v \in \text{supp}(\mu)} \mu(v) \cdot \kappa(I)(v)(E) \\ &= \sum_{v | f(v) \in \text{supp}(\mu)} \mu(f(v)) \cdot \kappa(I)(f(v))(E) && \text{(Because } f \text{ is bijective)} \\ &= \sum_{v \in \text{supp}(\mu')} \mu'(v) \cdot \kappa(I)(f(v))(E) && \text{(Because } \mu'(v) = \mu(f(v))) \\ &= \text{bind}(\mu', \lambda v. \kappa(I)(f(v)))(E) \end{aligned}$$

Thus, $\mu = \text{bind}(\mu', \lambda v. \kappa(I)(f(v)))$. Furthermore, $(K(f(v)))((\mathcal{F}, \kappa(I)(f(v)), p))$.

Thus, if we denote $\lambda v. \kappa(I)(f(v))$ as κ' , it satisfies

$$\begin{aligned} & (\mathcal{F}, \mu, p) \preceq a \wedge \forall i \in I. \mu(i) = \text{bind}(\mu', \kappa'(i)) \\ & \wedge \forall v \in \text{supp}(\mu). (K(v))((\mathcal{F}, \kappa'(I)(v), p)) \end{aligned}$$

Thus, $(C'_\mu v. K(f(v)))((\mathcal{F}, \mu, p))$. \square

LEMMA F.17. *Section 4.3 is sound.*

PROOF. Assume $a \in \mathcal{M}_I$ is a valid resource that satisfies $C_\mu v. (K(v) * \llbracket E \langle i \rangle \rrbracket)$. Then, by definition, we know that, for some $(\mathcal{F}_0, \mu_0, p_0)$ and κ_0 :

$$(\mathcal{F}_0, \mu_0, p_0) \preceq a \tag{49}$$

$$\forall i \in I. \mu_0(i) = \text{bind}(\mu, \kappa_0(i)) \tag{50}$$

and, for all $v \in \text{supp}(\mu)$, there are $(\mathcal{F}_1^v, \mu_1^v, p_1^v)$, $(\mathcal{F}_2^v, \mu_2^v, p_2^v)$ such that

$$(\mathcal{F}_1^v, \mu_1^v, p_1^v) \cdot (\mathcal{F}_2^v, \mu_2^v, p_2^v) \preceq (\mathcal{F}_0, \kappa_0(I)(v), p_0) \tag{51}$$

$$K(v)(\mathcal{F}_1^v, \mu_1^v, p_1^v) \tag{52}$$

$$\llbracket E \langle i \rangle \rrbracket (\mathcal{F}_2^v, \mu_2^v, p_2^v) \quad (53)$$

From (53) we know that for all $v \in \text{supp}(\mu)$ there are $L_1^v, L_0^v, U_1^v, U_0^v \in \mathcal{F}_2^v(i)$ such that:

$$\begin{aligned} L_0^v &\subseteq E^{-1}(\text{False}) \subseteq U_0^v & \mu_2^v(L_0^v) &= \mu_2^v(U_0^v) = 0 \\ L_1^v &\subseteq E^{-1}(\text{True}) \subseteq U_1^v & \mu_2^v(L_1^v) &= \mu_2^v(U_1^v) = 1 \end{aligned}$$

Without loss of generality, all $L_0^v, L_1^v, U_0^v, U_1^v$ can be assumed to be only non-trivial on $\text{pvar}(E)$. Consequently, we can also assume that $p_2^v(x \langle j \rangle) < 1$ for every $x \langle j \rangle$, and in addition $p_2^v(x \langle j \rangle) > 0$ if and only if $x \in \text{pvar } E$ and $j = i$. From these components we can construct a new resource:

$$\begin{aligned} \mathcal{F}_3(j) &\triangleq \begin{cases} \sigma\left(\left\{\bigcap_{v \in \text{supp}(\mu)} L_1^v, \bigcup_{v \in \text{supp}(\mu)} U_1^v\right\}\right) & \text{if } j = i \\ \{\mathbb{S}, \emptyset\} & \text{if } j \neq i \end{cases} \\ \mu_3 &\triangleq \mu_0|_{\mathcal{F}_3} \\ p_3 &\triangleq \lambda x \langle j \rangle. \begin{cases} \min\{p_2^v(x \langle i \rangle) \mid v \in \text{supp}(\mu)\} & \text{if } j = i \wedge x \in \text{pvar}(E) \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

By construction we obtain that $\forall j \in I. \mathcal{F}_3(j) \subseteq \mathcal{F}_0(j)$, and that $\mathcal{V}(\mathcal{F}_3, \mu_3, p_3)$. Now letting $p'_1 = p_0 - p_3$, we obtain a valid resource $(\mathcal{F}_0, \mu_0, p'_1)$.

Moreover, we have $\mathcal{F}_0 = \mathcal{F}_0 \oplus \mathcal{F}_3$ and $\forall j \in I. \forall X \in \mathcal{F}_3(j). \mu_3(X) \in \{0, 1\}$, which means that for any $X \in \mathcal{F}_3$ and $Y \in \mathcal{F}_0$, $\mu_3(X) \cdot \mu_0(Y) = \mu_0(X \cap Y)$. Then, by (50):

$$(\mathcal{F}_0, \text{bind}(\mu, \kappa_0), p'_1) \otimes (\mathcal{F}_3, \mu_3, p_3) \preceq (\mathcal{F}_0, \mu_0, p_0) = a$$

To close the proof it would then suffice to show that $C_\mu v. K(v)$ holds on $(\mathcal{F}_0, \text{bind}(\mu, \kappa_0), p'_1)$ and that $\llbracket E \langle i \rangle \rrbracket$ holds on $(\mathcal{F}_3(j), \mu_3, p_3)$. The latter is obvious. The former follows from the fact that $\kappa_0(j)(v)|_{\mathcal{F}_1^v} = \mu_1^v(j)$; by upward-closure and (52) this means that, for all $v \in \text{supp}(\mu)$:

$$K(v)(\mathcal{F}_1^v, \mu_1^v, p_1^v) \Rightarrow K(v)(\mathcal{F}_0, \kappa_0(I)(v), p'_1)$$

which proves our claim. \square

LEMMA F.18. *Section A is sound.*

PROOF. By unfolding the definitions,

$$\begin{aligned} &C_\mu v. \forall x : X. Q(v) \\ \Leftrightarrow &\exists \mathcal{F}, \mu_0, \kappa. \text{Own}((\mathcal{F}, \mu_0)) * \ulcorner \forall i \in I. \mu_0(i) = \text{bind}(\mu, \kappa(i)) \urcorner \\ &\quad * (\forall a \in A_\mu. \text{Own}((\mathcal{F}, [i: \kappa(i)(a) \mid i \in I])) -* \forall x : X. Q(v)) \\ \Rightarrow &\forall x : X. \exists \mathcal{F}, \mu_0, \kappa. \text{Own}((\mathcal{F}, \mu_0)) * \ulcorner \forall i \in I. \mu_0(i) = \text{bind}(\mu, \kappa(i)) \urcorner \\ &\quad * (\forall a \in A_\mu. \text{Own}((\mathcal{F}, [i: \kappa(i)(a) \mid i \in I])) -* Q(v)) \\ \Leftrightarrow &\forall x : X. C_\mu v. Q(v) \end{aligned}$$

\square

LEMMA F.19. *Section 4.3 is sound.*

PROOF. We first prove the forward direction: For any $a \in \mathcal{M}_I$, if $(\ulcorner \mu(X) = 1 \urcorner * C_\mu . K(v))((a))$, then there exists some $\mathcal{F}_0, \mu_0, p_0$, and κ_0 :

$$\begin{aligned} &(\mathcal{F}_0, \mu_0, p_0) \preceq a \\ &\forall i \in I. \mu_0(i) = \text{bind}(\mu, \kappa_0(i)) \\ &\forall v \in \text{supp}(\mu). (K(v))((\mathcal{F}_0, \kappa_0(I)(v), p_0)) \end{aligned}$$

The pure fact $\ulcorner \mu(X) = 1 \urcorner$ implies that $X \supseteq \text{supp}(\mu)$, and thus for every $v \in \text{supp}(\mu)$, $\ulcorner v \in X \urcorner$. Therefore, $(K(v))((\mathcal{F}_0, \kappa_0(I)(v), \mathbf{p}_0))$, which witnesses that $(C_\mu. \ulcorner v \in X \urcorner * K(v))((a))$.

We then prove the backward direction: if $C_\mu. \ulcorner v \in X \urcorner * K(v)$, then there exists $\mathcal{F}_0, \mu_0, \mathbf{p}_0$, and κ_0 :

$$\begin{aligned} (\mathcal{F}_0, \mu_0, \mathbf{p}_0) &\preceq a \\ \forall i \in I. \mu_0(i) &= \text{bind}(\mu, \kappa_0(i)) \\ \forall v \in \text{supp}(\mu). (\ulcorner v \in X \urcorner * K(v)) &((\mathcal{F}_0, \kappa_0(I)(v), \mathbf{p}_0)) \end{aligned}$$

Then it must $X \supseteq \text{supp}(\mu)$, which implies that $\ulcorner \mu(X) = 1 \urcorner$. Meanwhile, $\ulcorner v \in X \urcorner * K(v)$ holding on $(\mathcal{F}_0, \kappa_0(I)(v), \mathbf{p}_0)$ implies that $K(v)$ holds on $(\mathcal{F}_0, \kappa_0(I)(v), \mathbf{p}_0)$. Therefore, $\ulcorner \mu(X) = 1 \urcorner * C_\mu. K(v)$ holds on a . \square

F.2 Soundness of Primitive WP Rules

F.2.1 Structural Rules.

LEMMA F.20. *Section 2.4 is sound.*

PROOF. For any resource a , if $(\mathbf{wp} \, t \{Q\})(a)$, then

$$\forall \mu_0. \forall c. (a \cdot c) \preceq \mu_0 \Rightarrow \exists b. ((b \cdot c) \preceq \llbracket t \rrbracket(\mu_0) \wedge (Q)((b)))$$

From the premise $Q \vdash Q'$, and the fact that b must be valid for $(b \cdot c) \preceq \llbracket t \rrbracket(\mu_0)$ to hold, we have that $Q(b)$ implies $Q'(b)$. Thus, it must

$$\forall \mu_0. \forall c. (a \cdot c) \preceq \mu_0 \Rightarrow \exists b. ((b \cdot c) \preceq \llbracket t \rrbracket(\mu_0) \wedge Q'(b)),$$

which says $(\mathbf{wp} \, t \{Q'\})(a)$. \square

LEMMA F.21. *Section 2.4 is sound.*

PROOF. Let $a \in \mathcal{M}_I$ be a valid resource such that it satisfies $P * \mathbf{wp} \, t \{Q\}$. By definition, this means that, for some a_1, a_2 :

$$a_1 \cdot a_2 \preceq a \tag{54}$$

$$P(a_1) \tag{55}$$

$$\forall \mu_0, c. (a_2 \cdot c) \preceq \mu_0 \Rightarrow \exists b. ((b \cdot c) \preceq \llbracket t \rrbracket(\mu_0) \wedge Q(b)) \tag{56}$$

Our goal is to prove a satisfies $\mathbf{wp} \, t \{P * Q\}$, which, by unfolding the definitions, amounts to:

$$\exists a' \preceq a. \forall \mu_0, c'. (a' \cdot c') \preceq \mu_0 \Rightarrow \exists b_1, b. ((b_1 \cdot b) \cdot c') \preceq \llbracket t \rrbracket(\mu_0) \wedge P(b_1) \wedge Q(b) \tag{57}$$

Our goal can be proven by instantiating $a' = (a_1 \cdot a_2)$ and $b_1 = a_1$, from which we reduce the goal to proving, for all μ_0, c' :

$$((a_1 \cdot a_2) \cdot c') \preceq \mu_0 \Rightarrow \exists b. ((a_1 \cdot b) \cdot c') \preceq \llbracket t \rrbracket(\mu_0) \wedge P(a_1) \wedge Q(b) \tag{58}$$

We have that $P(a_1)$ holds by (55). By associativity and commutativity of the RA operation, we reduce the goal to:

$$(a_2 \cdot (a_1 \cdot c')) \preceq \mu_0 \Rightarrow \exists b. (b \cdot (a_1 \cdot c')) \preceq \llbracket t \rrbracket(\mu_0) \wedge Q(b) \tag{59}$$

This follows by applying assumption (56) with $c = (a_1 \cdot c')$. \square

LEMMA F.22. *Section 2.4 is sound.*

PROOF. Because $t_1 \cdot t_2$ is defined, it must $|t_1| \cap |t_2| = \emptyset$. By definition because $|t_1| \cap |t_2| = \emptyset$, we have $\llbracket (t_1 \cdot t_2) \rrbracket(\mu) = \llbracket t_2 \rrbracket(\llbracket t_1 \rrbracket(\mu))$.

- For the \vdash case: Assume a is a valid resource such that $(\mathbf{wp} \, t_1 \{ \mathbf{wp} \, t_2 \{ Q \} \})(a)$ holds. Our goal is to prove $(\mathbf{wp} \, t_1 \cdot t_2 \{ Q \})(a_0)$ holds, which unfolds by definition of WP into:

$$\forall \mu_0. \forall c_0. (a_0 \cdot c_0) \preceq \mu_0 \Rightarrow \exists a_2. ((a_2 \cdot c_0) \preceq \llbracket t_1 \cdot t_2 \rrbracket(\mu_0) \wedge Q(a_2)) \quad (60)$$

Take an arbitrary μ_0 and c_0 such that $(a_0 \cdot c_0) \preceq \mu_0$. By unfolding the WPs in the assumption, we have that there exists a $a_1 \in \mathcal{M}_I$ such that:

$$(a_1 \cdot c_0) \preceq \llbracket t_1 \rrbracket(\mu_0) \quad (61)$$

$$\forall \mu_1. \forall c_1. (a_1 \cdot c_1) \preceq \mu_1 \Rightarrow \exists a_2. (a_2 \cdot c_1) \preceq \llbracket t_2 \rrbracket(\mu_1) \wedge Q(a_2) \quad (62)$$

We can apply (62) to (61) by instantiating μ_1 with $\llbracket t_1 \rrbracket(\mu_0)$, and c_1 with c_0 , obtaining:

$$\exists a_2. ((a_2 \cdot c_0) \preceq \llbracket t_1 \rrbracket(\llbracket t_2 \rrbracket(\mu_0)) \wedge Q(a_2))$$

When $t_1 \cdot t_2$ is defined, the terms t_1 and t_2 are on disjoint indices, and thus $\llbracket t_1 \cdot t_2 \rrbracket(\mu_0) = \llbracket t_1 \rrbracket(\llbracket t_2 \rrbracket(\mu_0))$, we obtain the goal (60) as desired.

- For the \vdash case: For any resource a ,

$$\begin{aligned} & \mathbf{wp} \, (t_1 \cdot t_2) \{ Q \}(a) \\ \Leftrightarrow & \forall \mu_0. \forall c. (a \cdot c) \preceq \mu_0 \Rightarrow \exists b. ((b \cdot c) \preceq \llbracket t_1 \cdot t_2 \rrbracket(\mu_0) \wedge (Q)(b)) \end{aligned} \quad (63)$$

Since $(b \cdot c) \preceq \llbracket t_1 \cdot t_2 \rrbracket(\mu_0)$, we have $\mathcal{V}(b \cdot c)$ and thus $\mathcal{V}(b)$. Say

$$\begin{aligned} b &= [i: \mathcal{F}_b(i), \mu_b(i), p_b(i)] \\ c &= [i: \mathcal{F}_c(i), \mu_c(i), p_c(i)] \end{aligned}$$

Let

$$b' = \begin{cases} i: (\mathcal{F}_b(i), \mu_b(i), p_b(i)) & \text{if } i \in |t_1| \\ i: (\mathcal{F}(i), \mu(i), p(i)) & \text{if } i \notin |t_1| \end{cases}$$

Since $\mathcal{V}(b \cdot c)$ and $\mathcal{V}(a \cdot c)$, on each index $i \in I$, we have $\mathcal{V}(b'(i) \cdot c(i))$. Thus, $\mathcal{V}(b' \cdot c)$. Also, for each $i \in I$, $(b'(i) \cdot c(i)) \preceq \llbracket t_1(i) \rrbracket(\mu_0(i))$, which implies that

$$(b' \cdot c) \preceq \llbracket t_1 \rrbracket(\mu_0)$$

We want to show next that $(\mathbf{wp} \, t_2 \{ Q \})(b')$. For any $c' = [i: \mathcal{F}'_c(i), \mu'_c(i), p'_c(i)]$ such that $\mathcal{V}(b' \cdot c')$, it must

$$\begin{aligned} & \mathcal{V}((\mathcal{F}_b(i), \mu_b(i), p_b(i)) \cdot (\mathcal{F}'_c(i), \mu'_c(i), p'_c(i))) && \text{if } i \in |t_1| \\ & \mathcal{V}((\mathcal{F}(i), \mu(i), p(i)) \cdot (\mathcal{F}'_c(i), \mu'_c(i), p'_c(i))) && \text{if } i \in |t_2| \end{aligned}$$

By Eq. (63), $\mathcal{V}((\mathcal{F}(i), \mu(i), p(i)) \cdot (\mathcal{F}'_c(i), \mu'_c(i), p'_c(i)))$ also implies

$$\mathcal{V}((\mathcal{F}_b(i), \mu_b(i), p_b(i)) \cdot (\mathcal{F}'_c(i), \mu'_c(i), p'_c(i))).$$

Thus, $(b \cdot c) \preceq \llbracket t_1 \cdot t_2 \rrbracket(\mu_0) \wedge Q(b)$ witnesses that $\mathbf{wp} \, t_2 \{ Q \}(b')$.

□

LEMMA F.23. *Section A is sound.*

PROOF. For any resource a ,

$$\begin{aligned} & (\mathbf{wp} \, t_1 \{ Q_1 \} \wedge \mathbf{wp} \, t_2 \{ Q_2 \})(a) \\ \Leftrightarrow & \forall \mu_0. \forall c. (a \cdot c) \preceq \mu_0 \Rightarrow \\ & \exists b. ((b \cdot c) \preceq \llbracket t_1 \rrbracket(\mu_0) \wedge Q(b)) \wedge \exists b'. ((b' \cdot c) \preceq \llbracket t_2 \rrbracket(\mu_0) \wedge Q(b')) \end{aligned}$$

Define b'' such that

$$b''(i) = \begin{cases} b(i) & \text{if } i \in \text{idx}(Q_1) \\ b'(i) & \text{otherwise} \end{cases}$$

For any c , $\mathcal{V}(a \cdot c)$ implies $\mathcal{V}(b'' \cdot c)$ because $\mathcal{V}(b'(i) \cdot c(i))$ and $\mathcal{V}(b(i) \cdot c(i))$ for all i . Furthermore, $b''(i) = b(i)$ for $i \in \text{idx}(Q_1)$ implies that $Q_1(b'')$. Also, $\text{idx}(Q_2) \cap |t_1| \subseteq |t_2|$ implies that $Q_2(b'')$. Therefore, $(Q_1 \wedge Q_2)(b'')$, witnessing $(\mathbf{wp} \, t_1 + t_2 \{Q_1 \wedge Q_2\})(a)$. \square

LEMMA F.24. Section 5.1 is sound.

PROOF. By the meaning of conditioning modality and weakest precondition transformer,

$$\begin{aligned} & (\text{own}_{\mathbb{X}} \wedge C_{\mu} v. \mathbf{wp} \, t \{Q(v)\})(a) \\ \Leftrightarrow & \text{own}_{\mathbb{X}}(a) \wedge \exists \mathcal{F}, \mu, p, \kappa. (\mathcal{F}, \mu, p) \preceq a \wedge \forall i \in I. \mu(i) = \text{bind}(\mu, \kappa(i)) \\ & \wedge \forall v \in \text{supp}(\mu). (\mathbf{wp} \, t \{Q(v)\})(\mathcal{F}, \kappa(I)(v), p) \end{aligned}$$

Intuitively, for each v , running t on each fibre $(\mathcal{F}, \kappa(I)(v), p)$ gives a output resource that satisfies $Q(v)$.

Assume $\mathcal{V}(a)$ holds and let $a = (\mathcal{F}_a, \mu_a, p_a)$. By Lemma C.4, when $(\mathcal{F}, \mu, p) \preceq a$, $\mu = \text{bind}(\mu, \kappa)$ iff that there exists κ'' such that $\mu_a = \text{bind}(\mu, \kappa'')$ and $\kappa(I)(v) \sqsubseteq \kappa''(I)(v)$ for every v . Thus,

$$\begin{aligned} (C_{\mu} v. \mathbf{wp} \, t \{Q(v)\})(\mathcal{F}_a, \mu_a, p_a) \Leftrightarrow & \exists \kappa. \forall i \in I. \mu_a(i) = \text{bind}(\mu, \kappa''(i)) \\ & \wedge \forall v \in \text{supp}(\mu). (\mathbf{wp} \, t \{Q(v)\})(\mathcal{F}, \kappa(I)(v), p) \end{aligned}$$

We want to show that

$$\mathbf{wp} \, t \{C_{\mu} v. Q(v)\}(a)$$

which is equivalent to

$$\forall \mu'. \forall c. a \cdot c \preceq \mu' \Rightarrow \exists a'. a' \cdot c \preceq \llbracket t \rrbracket(\mu') \wedge (C_{\mu} Q(v))(a).$$

Let's fix an arbitrary μ', c that satisfy $\mathcal{V}(a \cdot c) \wedge a \cdot c \preceq a_{\mu'}$, we try to construct a corresponding a' . The high-level approach that we will take is to show that running t on a takes us to a resource that is equivalent to bind the set of output resource satisfying $Q(v)$ to μ .

Recall that $a = (\mathcal{F}_a, \mu_a, p_a)$ also satisfies $\text{own}_{\mathbb{X}}$, which says $\mathcal{F}_a = \Sigma_{\mathbb{X}}$. We claim that $a \cdot c \preceq (\Sigma_{\mathbb{X}}, \mu', p_1)$ holds implies that the probability space c is trivial. Say $c = (\mathcal{F}_c, \mu_c, p_c)$, then for any $E \in \mathcal{F}_c$, the event E must also in \mathcal{F}_a and $\Sigma_{\mathbb{X}}$ because they are the full sigma algebra. By definition of $a \cdot c \preceq (\Sigma_{\mathbb{X}}, \mu', p_1)$, we have

$$\mu_c(E) \cdot \mu_a(E) = \mu'(E \cap E) = \mu'(E). \quad (64)$$

Another implication of $a \cdot c \preceq (\Sigma_{\mathbb{X}}, \mu', p_1)$ is that we have $\mu_c(E) = \mu'(E)$ and $\mu_a(E) = \mu'(E)$. Combining with Eq. (64), we can conclude

$$\mu'(E) \cdot \mu'(E) = \mu'(E),$$

which implies that $\mu_c(E) = \mu'(E) \in \{0, 1\}$. Therefore, c is a trivial probability space and

$$(\mathcal{F}_a, \kappa(I)(v), p_a) \cdot c \preceq (\mathcal{F}_a, \kappa(I)(v), p_a)$$

Furthermore, for every $v \in \text{supp}(\mu)$, we have $(\mathbf{wp} \, t \{Q(v)\})(\mathcal{F}, \kappa(I)(v), p)$ which implies

$$\forall \kappa'. (\mathcal{F}_a, \kappa(I)(v), p_a) \cdot c \preceq \kappa'(I)(v) \quad (65)$$

$$\Rightarrow \exists a_v. (a_v \cdot c \preceq \llbracket t \rrbracket(\kappa'(I)(v))) \wedge Q(v)(a_v). \quad (66)$$

Therefore,

$$a \cdot c \preceq a_{\mu'} \Rightarrow \forall v \in \text{supp}(\mu). (\mathcal{V}((\mathcal{F}_a, \kappa(I)(v), p_a) \cdot c) \wedge (\mathcal{F}_a, \kappa(I)(v), p_a) \cdot c \preceq (\Sigma_{\mathbb{X}}, \kappa(I)(v), 1))$$

(By C.7 and C.4)

$$\begin{aligned} &\Rightarrow \forall v \in \text{supp}(\mu). \exists a_v. \mathcal{V}(a_v \cdot c) \wedge (a_v \cdot c \preceq (\Sigma_{\mathbb{X}}, \llbracket t \rrbracket(\kappa(I)(v)), 1)) \wedge Q(v)(a_v) && \text{(By Eq. (65))} \\ &\Rightarrow \forall v \in \text{supp}(\mu). \mathbf{p}_{a_v} + \mathbf{p}_c \preceq 1 \wedge Q(v)(\Sigma_{\mathbb{X}}, \llbracket t \rrbracket(\kappa'(I)(v)), 1). && \text{(By upwards closure)} \end{aligned}$$

Let $a'_v = (\Sigma_{\mathbb{X}}, \llbracket t \rrbracket(\kappa'(I)(v)), \mathbf{p}_a)$. Because $\mu_c(E) \in \{0, 1\}$ for any $E \in \mathcal{F}_c$, for every v , we have $(\Sigma_{\mathbb{X}}, \llbracket t \rrbracket(\kappa'(I)(v))) \cdot (\mathcal{F}_c, \mu_c)$ defined and thus $a'_v \cdot c$ valid. Define

$$a' = (\Sigma_{\mathbb{X}}, \text{bind}(\mu, \lambda v. \llbracket t \rrbracket(\kappa'(I)(v)), \mathbf{p}_a)$$

By Lemma C.7, $\mathcal{V}(a'_v \cdot c)$ for all $v \in \text{supp}_\mu$ implies $\mathcal{V}(a' \cdot c)$. Also, because $Q(v)(a_v)$ for all $v \in A_\mu$, $(C_\mu v. Q(v))(a')$. Thus, $(\mathbf{wp} \, t \{C_\mu v. Q(v)\})(a)$. \square

F.2.2 Program Rules.

LEMMA F.25. *Section A is sound.*

PROOF. Assume $a \in \mathcal{M}_I$ is valid and such that $P(a)$ holds. By unfolding the definition of WP, we need to prove

$$\forall \mu_0. \forall c. (a \cdot c) \preceq \mu_0 \Rightarrow \exists b. ((b \cdot c) \preceq \llbracket t \rrbracket(\mu_0) \wedge P(b))$$

which follows trivially by $\llbracket [i: \text{skip}] \rrbracket(\mu_0) = \mu_0$ and picking $b = a$. \square

LEMMA F.26. *Section 2.4 is sound.*

PROOF. Assume $a_0 \in \mathcal{M}_I$ is a valid resource such that $(\mathbf{wp} \, [i: t] \{ \mathbf{wp} \, [i: t'] \{ Q \} \})(a_0)$ holds. Our goal is to prove $(\mathbf{wp} \, ([i: t; t']) \{ Q \})(a_0)$ holds, which unfolds by definition of WP into:

$$\forall \mu_0. \forall c_0. (a_0 \cdot c_0) \preceq \mu_0 \Rightarrow \exists a_2. ((a_2 \cdot c_0) \preceq \llbracket [i: t; t'] \rrbracket(\mu_0) \wedge Q(a_2)) \quad (67)$$

Take an arbitrary μ_0 and c_0 such that $(a_0 \cdot c_0) \preceq \mu_0$. By unfolding the WPs in the assumption, we have that there exists a $a_1 \in \mathcal{M}_I$ such that:

$$(a_1 \cdot c_0) \preceq \llbracket [i: t] \rrbracket(\mu_0) \quad (68)$$

$$\forall \mu_1. \forall c_1. (a_1 \cdot c_1) \preceq \mu_1 \Rightarrow \exists a_2. ((a_2 \cdot c_1) \preceq \llbracket [i: t'] \rrbracket(\mu_1) \wedge Q(a_2)) \quad (69)$$

We can apply (69) to (68) by instantiating μ_1 with $\llbracket [i: t] \rrbracket(\mu_0)$, and c_1 with c_0 , obtaining:

$$\exists a_2. ((a_2 \cdot c_0) \preceq \llbracket [i: t'] \rrbracket(\llbracket [i: t] \rrbracket(\mu_0)) \wedge Q(a_2))$$

Since by definition, $\llbracket [i: t'] \rrbracket(\mu_0) = \llbracket [i: t'] \rrbracket(\llbracket [i: t] \rrbracket(\mu_0))$, we obtain the goal (67) as desired. \square

LEMMA F.27. *Section 4.4 is sound.*

PROOF. Let $a \in \mathcal{M}_I$ be a valid resource, and let $a(i) = (\mathcal{F}, \mu, p)$. By assumption we have $p(x) = 1$ and $p(y) > 0$ for all $y \in \text{pvar}(e)$. We want to show that a satisfies $\mathbf{wp} \, [i: x := e] \{ \llbracket x \langle i \rangle = e \langle i \rangle \rrbracket \}$. This is equivalent to

$$\forall \mu_0. \forall c. (a \cdot c \preceq \mu_0) \Rightarrow \exists b. (b \cdot c \preceq \llbracket [i: x := e] \rrbracket(\mu_0) \wedge \llbracket x \langle i \rangle = e \langle i \rangle \rrbracket(b))$$

We show this holds by picking b as follows:

$$b \triangleq a[i: (\mathcal{F}_b, \mu_b, p)] \quad \mathcal{F}_b \triangleq \{\mathbb{S}, \emptyset, A, \mathbb{S} \setminus A\} \quad A \triangleq \{s[x \mapsto \llbracket e \rrbracket(s)] \mid s \in \mathbb{S}\}$$

where μ_b is determined by setting $\mu_b(A) = 1$.

By construction we have that $\llbracket x \langle i \rangle = e \langle i \rangle \rrbracket(b)$ holds. To close the proof we then need to show that $(b \cdot c) \preceq \llbracket [i: x := e] \rrbracket(\mu_0)$.

Let $c(i) = (\mathcal{F}_c, \mu_c, p_c)$. Observe that by the assumptions on p , we have $\mathcal{V}(b)$ since \mathcal{F}_b is only non-trivial on $\text{pvar}(e) \cup \{x\}$; moreover, by the assumption $\mathcal{V}(a \cdot c)$ we have that $\mathcal{V}(p + p_c)$ holds, which means that $p_c(x) = 0$, and thus \mathcal{F}_c is trivial on x .

Let us define the function $\text{pre}: \wp(\mathbb{S}) \rightarrow \wp(\mathbb{S})$ as:

$$\text{pre}(X) \triangleq \{s \mid s[x \mapsto \llbracket e \rrbracket(s)] \in X\}.$$

That is, $\text{pre}(X)$ is the weakest precondition (in the standard sense) of the assignment. By construction, we have:

$$\begin{aligned} \text{pre}(A) &= \mathbb{S} & \text{pre}(X_1 \cap X_2) &= \text{pre}(X_1) \cap \text{pre}(X_2) \\ \text{pre}(\mathbb{S} \setminus A) &= \emptyset & \text{pre}(X_c) &= X_c \text{ for all } X_c \in \mathcal{F}_c \end{aligned}$$

In particular, the latter holds because \mathcal{F}_c is trivial in x .

By unfolding the definition of $\llbracket \cdot \rrbracket$, it is easy to check that for every $X \in \Sigma_{\mathbb{S}}$:

$$\llbracket x := e \rrbracket(\mu_0)(X) = \mu_0(\text{pre}(X))$$

We are now ready to show $(b \cdot c) \preceq \llbracket [i: x := e] \rrbracket(\mu_0)$ by showing that $(\mathcal{F}_b, \mu_b) \otimes (\mathcal{F}_c, \mu_c) = (\mathcal{F}_b \oplus \mathcal{F}_c, \llbracket x := e \rrbracket(\mu_0)|_{(\mathcal{F}_b \oplus \mathcal{F}_c)})$ where $\mu_0 = \mu_0(i)$. To show this it suffices to prove that for every $X_b \in \mathcal{F}_b$ and every $X_c \in \mathcal{F}_c$, $\llbracket x := e \rrbracket(\mu_0)(X_b \cap X_c) = \mu_b(X_b) \cdot \mu_c(X_c)$. We proceed by case analysis on X_b :

– **Case** $X_b = A$. Then:

$$\begin{aligned} \llbracket x := e \rrbracket(\mu_0)(A \cap X_c) &= \mu_0(\text{pre}(A \cap X_c)) \\ &= \mu_0(\text{pre}(A) \cap \text{pre}(X_c)) \\ &= \mu_0(\mathbb{S} \cap \text{pre}(X_c)) \\ &= \mu_0(\text{pre}(X_c)) \\ &= \mu_b(A) \cdot \mu_c(X_c) \\ &= \mu_b(A) \cdot \mu_c(X_c) \end{aligned}$$

– **Case** $X_b = \mathbb{S} \setminus A$. Then:

$$\begin{aligned} \llbracket x := e \rrbracket(\mu_0)(\mathbb{S} \setminus A \cap X_c) &= \mu_0(\text{pre}((\mathbb{S} \setminus A) \cap X_c)) \\ &= \mu_0(\text{pre}(\mathbb{S} \setminus A) \cap \text{pre}(X_c)) \\ &= \mu_0(\emptyset \cap \text{pre}(X_c)) \\ &= 0 \\ &= \mu_b(\mathbb{S} \setminus A) \cdot \mu_c(X_c) \end{aligned}$$

– **Case** $X_b = \mathbb{S}$ or $X_b = \emptyset$. Analogous to the previous cases. \square

LEMMA F.28. *Section 4.4 is sound.*

PROOF. Assume $a \in \mathcal{M}_I$ is valid and such that $a(i) = (\mathcal{F}, \mu, p)$, with $p(x) = 1$. Our goal is to show that a satisfies $\mathbf{wp} [i: x \approx d(\vec{v})] \{x \langle i \rangle \sim d(\vec{v})\}$ which is equivalent to proving, for all μ_0 and for all c :

$$(a \cdot c \preceq \mu_0) \Rightarrow \exists b. (b \cdot c \preceq \llbracket [i: x \approx d(\vec{v})] \rrbracket(\mu_0) \wedge (x \sim d(\vec{v}))(b)) \quad (70)$$

Let $\mu_0 = \mu_0(i)$ and $\mu_1 = \llbracket x \approx d(\vec{v}) \rrbracket(\mu_0)$. Moreover, let $c(i) = (\mathcal{F}_c, \mu_c, p_c)$. Observe that by the assumptions on p and validity of $a \cdot c$, we have $p_c(x) = 0$, which means \mathcal{F}_c is trivial on x . We aim to prove (70) by letting

$$\begin{aligned} b &\triangleq a[i: (\mathcal{F}_b, \mu_b, p_b)] & \mu_b &\triangleq \mu_1|_{\mathcal{F}_b} \\ \mathcal{F}_b &\triangleq \sigma(\{ \{s \in \mathbb{S} \mid s(x) = v\} \mid v \in \mathbb{V} \}) & p_b &\triangleq (x:1) \end{aligned}$$

Note that by construction $\mathcal{V}(p_b + p_c)$, and $\mathcal{V}(b)$ since \mathcal{F}_b is only non-trivial in x . Similarly to the proof for Section 4.4, we define the function $\text{pre}: \wp(\mathbb{S}) \rightarrow \wp(\mathbb{S})$ as:

$$\text{pre}(X) \triangleq \{s \mid \exists v \in \mathbb{V}. s[x \mapsto v] \in X\}.$$

Since \mathcal{F}_c is trivial on x , for all $X_c \in \mathcal{F}_c$, $\text{pre}(X_c) = X_c$. Moreover, for all $X_b \in \mathcal{F}_b \setminus \{\emptyset\}$, $\text{pre}(X_b) = \mathbb{S}$, since X_b is trivial on every variable except x .

By unfolding the definitions, we have:

$$\begin{aligned} \mu_1(X) &= \llbracket x \approx d(\vec{v}) \rrbracket(\mu_0)(X) \\ &= \sum_{s \in X} \mu_0(\text{pre}(s)) \cdot \llbracket d \rrbracket(\vec{v})(s(x)) \end{aligned}$$

We now show that $(\mathcal{F}_b, \mu_b) \otimes (\mathcal{F}_c, \mu_c) = (\mathcal{F}_b \oplus \mathcal{F}_c, \mu_1|_{(\mathcal{F}_b \oplus \mathcal{F}_c)})$ by showing that for all $X_b \in \mathcal{F}_b$ and $X_c \in \mathcal{F}_c$: $\mu_1(X_b \cap X_c) = \mu_b(X_b) \cdot \mu_c(X_c)$. To prove this we first define $V: \wp(\mathbb{S}) \rightarrow \wp(\mathbb{V})$ as $V(X) \triangleq \{s(x) \mid s \in X\}$, and $S_w \triangleq \{s \mid s(x) = w\}$. We observe that $X_b = \biguplus_{w \in V(X_b)} S_w$, and thus $X_b \cap X_c = \biguplus_{w \in V(X_b)} (X_c \cap S_w)$; moreover, $\text{pre}(X_c \cap S_w) = \{s \mid s[x \mapsto w] \in X_c\} = X_c$. Thus, we can calculate:

$$\begin{aligned} \mu_1(X_b \cap X_c) &= \sum_{s \in X_b \cap X_c} \mu_0(\text{pre}(s)) \cdot \llbracket d \rrbracket(\vec{v})(s(x)) \\ &= \sum_{w \in V(X_b)} \sum_{s \in X_c \cap S_w} \mu_0(\text{pre}(s)) \cdot \llbracket d \rrbracket(\vec{v})(w) \\ &= \sum_{w \in V(X_b)} \left(\llbracket d \rrbracket(\vec{v})(w) \cdot \sum_{s \in X_c \cap S_w} \mu_0(\text{pre}(s)) \right) \\ &= \left(\sum_{w \in V(X_b)} \llbracket d \rrbracket(\vec{v})(w) \cdot \mu_0(\text{pre}(X_c \cap S_w)) \right) \\ &= \left(\sum_{w \in V(X_b)} \llbracket d \rrbracket(\vec{v})(w) \right) \cdot \mu_0(X_c) \\ &= \mu_b(X_b) \cdot \mu_c(X_c) \end{aligned}$$

The last equation is given by $a \cdot c \preceq \mu_0$ which implies that $\mu_c = \mu_0|_{\mathcal{F}_c}$, and by:

$$\begin{aligned} \mu_b(X_b) &= \mu_1(X_b) = \sum_{s \in X_b} \mu_0(\text{pre}(s)) \cdot \llbracket d \rrbracket(\vec{v})(s(x)) \\ &= \sum_{w \in V(X_b)} \sum_{s \in S_w} \mu_0(\text{pre}(s)) \cdot \llbracket d \rrbracket(\vec{v})(w) \\ &= \sum_{w \in V(X_b)} \llbracket d \rrbracket(\vec{v})(w) \end{aligned}$$

Finally, we need to show $(x \sim d(\vec{v}))(b)$ which amounts to proving $x \prec (\mathcal{F}_b, \mu_b)$ and $\llbracket d \rrbracket(\vec{v}) = \mu_b \circ x^{-1}$. The former holds because by construction x is measurable in \mathcal{F}_b . For the latter, for all $W \subseteq \mathbb{V}$:

$$(\mu_b \circ x^{-1})(W) = \mu_b(x^{-1}(W)) = \sum_{w \in V(x^{-1}(W))} \llbracket d \rrbracket(\vec{v})(w) = \sum_{w \in W} \llbracket d \rrbracket(\vec{v})(w) = \llbracket d \rrbracket(\vec{v})(W). \quad \square$$

LEMMA F.29. Section A is sound.

PROOF. For any valid resource a ,

$$\begin{aligned}
& (\text{if } v \text{ then } \mathbf{wp} [i : t_1] \{Q(1)\} \text{ else } \mathbf{wp} [i : t_2] \{Q(0)\})(a) \\
& \Leftrightarrow \begin{cases} (\mathbf{wp} [i : t_1] \{Q(1)\})(a) & \text{if } v \doteq 1 \\ (\mathbf{wp} [i : t_2] \{Q(0)\})(a) & \text{otherwise} \end{cases} \\
& \Leftrightarrow \forall \mu_0. \forall c. (a \cdot c) \preceq \mu_0 \Rightarrow \begin{cases} \exists b. (b \cdot c) \preceq \llbracket i : t_1 \rrbracket(\mu_0) \wedge Q(1)(b) & \text{if } v \doteq 1 \\ \exists b. (b \cdot c) \preceq \llbracket i : t_2 \rrbracket(\mu_0) \wedge Q(0)(b) & \text{otherwise} \end{cases} \\
& \Leftrightarrow \forall \mu_0. \forall c. (a \cdot c) \preceq \mu_0 \Rightarrow \exists b. (b \cdot c) \preceq \llbracket i : \text{if } v \text{ then } t_1 \text{ else } t_2 \rrbracket(\mu_0) \wedge Q(v \doteq 1)(b) \\
& \Rightarrow \mathbf{wp} [i : \text{if } v \text{ then } t_1 \text{ else } t_2] \{Q(v \doteq 1)\}(a)
\end{aligned}$$

□

LEMMA F.30. *Section A is sound.*

PROOF. For any resource $a = (\mathcal{F}, \mu, p)$, $(\llbracket e \langle i \rangle = v \rrbracket * \mathbf{wp} [i : \mathcal{E}[v]] \{Q\})(\mathcal{F}, \mu, p)$ iff there exists $(\mathcal{F}_1, \mu_1, p_1), (\mathcal{F}_2, \mu_2, p_2)$ such that

$$\begin{aligned}
& (\llbracket e \langle i \rangle = v \rrbracket)(\mathcal{F}_1, \mu_1, p_1) \\
& (\mathbf{wp} [i : \mathcal{E}[v]] \{Q\})(\mathcal{F}_2, \mu_2, p_2) \\
& (\mathcal{F}_1, \mu_1, p_1) \cdot (\mathcal{F}_2, \mu_2, p_2) \preceq (\mathcal{F}, \mu, p)
\end{aligned}$$

By the upwards closure, we also have

$$\begin{aligned}
& (\llbracket e \langle i \rangle = v \rrbracket)(\mathcal{F}, \mu, p) \\
& (\mathbf{wp} [i : \mathcal{E}[v]] \{Q\})(\mathcal{F}, \mu, p)
\end{aligned}$$

The fact that $(\llbracket e \langle i \rangle = v \rrbracket)(\mathcal{F}_1, \mu_1, p_1)$ implies that $\mu_1((e \langle i \rangle = v)^{-1}(\text{True})) = 1$, which implies that $\llbracket e \rrbracket(s) = v$ for all $s \in \text{supp}(\mu_1(i))$.

By Lemma B.6, we have for any $s \in \mathbb{S}$,

$$\mathcal{K}[\llbracket \mathcal{E}[e] \rrbracket](s) = \mathcal{K}[\llbracket \mathcal{E}[\llbracket e \rrbracket(s) \rrbracket]](s),$$

which implies that for any μ_0 over $\Sigma_{\mathbb{S}}$

$$\begin{aligned}
\llbracket \mathcal{E}[e] \rrbracket(\mu_0) &= s \leftarrow \mu_0; \mathcal{K}[\llbracket \mathcal{E}[e] \rrbracket](s) \\
&= s \leftarrow \mu_0; \mathcal{K}[\llbracket \mathcal{E}[\llbracket e \rrbracket(s) \rrbracket]](s) \\
&= s \leftarrow \mu_0; \mathcal{K}[\llbracket \mathcal{E}[v] \rrbracket](s) \\
&= \llbracket \mathcal{E}[v] \rrbracket(\mu_0).
\end{aligned}$$

Define $\mu'_0 = \llbracket [i : \mathcal{E}[v]] \rrbracket \mu_0$. Thus, $(\mathbf{wp} [i : \mathcal{E}[v]] \{Q\})(a)$ iff

$$\forall \mu_0. \forall c. (\mathcal{V}(a \cdot c) \wedge a \cdot c \preceq a_{\mu_0}) \Rightarrow \exists a'. (\mathcal{V}(a' \cdot c) \wedge a' \cdot c \preceq a_{\mu'_0} \wedge Q(a'))$$

iff

$$\forall \mu_0. \forall c. (\mathcal{V}(a \cdot c) \wedge a \cdot c \preceq a_{\mu_0}) \Rightarrow \exists a'. (\mathcal{V}(a' \cdot c) \wedge a' \cdot c \preceq a_{\mu'_0} \wedge Q(a'))$$

iff $(\mathbf{wp} [i : \mathcal{E}[e]] \{Q\})(a)$.

□

LEMMA F.31. *Section 5.6 is sound.*

PROOF. By definition,

$$\begin{aligned} \llbracket \text{repeat } (n+1) \ t \rrbracket(\mu) &= (s \leftarrow \mu; s' \leftarrow \text{loop}_t(n, s); \mathcal{K} \llbracket t \rrbracket(s')) \\ &= \llbracket (\text{repeat } n \ t); t \rrbracket(\mu) \end{aligned}$$

thus the rule follows from the argument of Lemma F.26. \square

LEMMA F.32. *Section 5.6 is sound.*

PROOF. By induction on n .

- **Base case** $n = 0$. Analogously to Lemma F.25 since, by definition, $\llbracket \text{repeat } 0 \ t \rrbracket(\mu_0) = \mu_0$.
- **Induction step** $n > 0$. By induction hypothesis $P(0) \vdash \text{wp } [j: \text{repeat } (n-1) \ t] \{P(n-1)\}$ holds, and we want to show that $P(0) \vdash \text{wp } [j: \text{repeat } n \ t] \{P(n)\}$. By Lemma F.31, it suffices to show $P(0) \vdash \text{wp } [j: \text{repeat } (n-1) \ t] \{\text{wp } [j: t] \{P(n)\}\}$. By applying the induction hypothesis and Lemma F.20 we are left with proving $P(n-1) \vdash \text{wp } [j: t] \{P(n)\}$ which is implied by the premise of the rule with $i = n-1 < n$. \square

F.3 Soundness of Derived Rules

In this section we provide derivations for the rules we claim are derivable in BLUEBELL.

F.3.1 Ownership and Distributions.

LEMMA F.33. *Section A is sound.*

PROOF.

$$\begin{aligned} E\langle i \rangle \sim \delta_v \Vdash \exists \mathcal{F}, \mu. \text{Own}((\mathcal{F}, \mu)) * \ulcorner \mu \circ E\langle i \rangle^{-1} = \delta_v \urcorner \\ \Vdash \exists \mathcal{F}, \mu. \text{Own}((\mathcal{F}, \mu)) * \ulcorner \mu \circ (E\langle i \rangle = v)^{-1} = \delta_{\text{True}} \urcorner \\ \Vdash [E\langle i \rangle = v] \end{aligned} \quad \square$$

LEMMA F.34. *Section A is sound.*

PROOF.

$$\begin{aligned} [E\langle i \rangle = v] * [E\langle i \rangle = v'] \vdash E\langle i \rangle \sim \delta_v * E\langle i \rangle \sim \delta_{v'} & \quad (\text{SURE-DIRAC}) \\ \vdash E\langle i \rangle \sim \delta_v \wedge E\langle i \rangle \sim \delta_{v'} \\ \vdash \ulcorner \delta_v = \delta_{v'} \urcorner & \quad (\text{DIST-INJ}) \\ \vdash \ulcorner v = v' \urcorner & \quad \square \end{aligned}$$

LEMMA F.35. *Section A is sound.*

PROOF.

$$\begin{aligned} E_1\langle i \rangle \sim \mu * \ulcorner (E_2 = f(E_1))\langle i \rangle \urcorner \vdash C_\mu v. [E_1\langle i \rangle = v] * \ulcorner (E_2 = f(E_1))\langle i \rangle \urcorner & \quad (\text{C-UNIT-R, C-FRAME}) \\ \vdash C_\mu v. [E_1\langle i \rangle = v \wedge E_2\langle i \rangle = f(E_1\langle i \rangle)] & \quad (\text{SURE-MERGE}) \\ \vdash C_\mu v. [E_2\langle i \rangle = f(v)] & \quad (\text{C-CONS}) \\ \vdash C_\mu v. C_{\delta_{f(v)}} v'. [E_2\langle i \rangle = v'] & \quad (\text{C-UNIT-L}) \\ \vdash C_{\mu'} v'. [E_2\langle i \rangle = v'] & \quad (\text{C-ASSOC, C-SURE-PROJ}) \end{aligned}$$

where $\mu' = \text{bind}(\mu, \lambda x. \delta_{f(x)}) = \mu \circ f^{-1}$. By C-UNIT-R we thus get $E_2\langle i \rangle \sim \mu \circ f^{-1}$. \square

LEMMA F.36. *Section A is sound.*

PROOF. Assume $E: \mathbb{S} \rightarrow A$ and $f: A \rightarrow B$, then:

$$\begin{aligned}
E\langle i \rangle \sim \mu &\vdash C_\mu v. \llbracket (E = v)\langle i \rangle \rrbracket && \text{(C-UNIT-R)} \\
&\vdash C_\mu v. \llbracket (f \circ E)\langle i \rangle = f(v) \rrbracket && \text{(C-CONS)} \\
&\vdash C_\mu v. C_{\delta_{f(v)}} v'. \llbracket (f \circ E)\langle i \rangle = v' \rrbracket && \text{(C-UNIT-L)} \\
&\vdash C_{\mu'} v'. \llbracket (f \circ E)\langle i \rangle = v' \rrbracket && \text{(C-ASSOC, C-SURE-PROJ)}
\end{aligned}$$

where $\mu' = \text{bind}(\mu, \lambda x. \delta_{f(x)}) = \mu \circ f^{-1}$. By C-UNIT-R we thus get $(f \circ E)\langle i \rangle \sim \mu \circ f^{-1}$. \square

LEMMA F.37. *Section A is sound.*

PROOF.

$$\begin{aligned}
E\langle i \rangle \sim \delta_v &\vdash \llbracket E\langle i \rangle = v \rrbracket && \text{(SURE-DIRAC)} \\
&\vdash \llbracket E\langle i \rangle = v \rrbracket * \llbracket E\langle i \rangle = v \rrbracket && \text{(SURE-MERGE)} \\
&\vdash E\langle i \rangle \sim \delta_v * E\langle i \rangle \sim \delta_v && \text{(SURE-DIRAC)}
\end{aligned}$$

\square

LEMMA F.38. *Section A is sound.*

PROOF.

$$\begin{aligned}
E\langle i \rangle \sim \mu &\vdash C_\mu v. \llbracket E\langle i \rangle = v \rrbracket && \text{(C-UNIT-R)} \\
&\vdash \ulcorner \mu(\text{supp}(\mu)) = 1 \urcorner * C_\mu v. \llbracket E\langle i \rangle = v \rrbracket && \\
&\vdash C_\mu v. (\ulcorner v \in \text{supp}(\mu) \urcorner * \llbracket E\langle i \rangle = v \rrbracket) && \text{(C-PURE)} \\
&\vdash C_\mu v. (\llbracket E\langle i \rangle = v \rrbracket * \llbracket E\langle i \rangle \in \text{supp}(\mu) \rrbracket) && \\
&\vdash (C_\mu v. \llbracket E\langle i \rangle = v \rrbracket) * \llbracket E\langle i \rangle \in \text{supp}(\mu) \rrbracket && \text{(SURE-STR-CONVEX)} \\
&\vdash E\langle i \rangle \sim \mu * \llbracket E\langle i \rangle \in \text{supp}(\mu) \rrbracket && \text{(C-UNIT-R)}
\end{aligned}$$

\square

LEMMA F.39. *Section A is sound.*

PROOF.

$$\begin{aligned}
E_1\langle i \rangle \sim \mu_1 * E_2\langle i \rangle \sim \mu_2 &\vdash C_{\mu_1} v_1. C_{\mu_2} v_2. (\llbracket E_1\langle i \rangle = v_1 \rrbracket * \llbracket E_2\langle i \rangle = v_2 \rrbracket) && \text{(C-UNIT-R, C-FRAME)} \\
&\vdash C_{\mu_1} v_1. C_{\mu_2} v_2. \llbracket (E_1, E_2)\langle i \rangle = (v_1, v_2) \rrbracket && \text{(SURE-MERGE)} \\
&\vdash C_{\mu_1 \otimes \mu_2} (v_1, v_2). \llbracket (E_1, E_2)\langle i \rangle = (v_1, v_2) \rrbracket && \text{(C-ASSOC)} \\
&\vdash (E_1\langle i \rangle, E_2\langle i \rangle) \sim \mu_1 \otimes \mu_2 && \text{(C-UNIT-R)}
\end{aligned}$$

\square

F.3.2 Joint conditioning.

LEMMA F.40. *Section 2.3 is sound.*

PROOF. Recall that $\mu \prec \kappa \triangleq \lambda(v, w). \mu(v) \kappa(v)(w)$. which can be reformulated as $\mu \prec \kappa = \text{bind}(\mu, \lambda v. (\text{bind}(\kappa(v), \lambda w. \text{return}(v, w))))$.

The (\vdash) direction is an instance of C-ASSOC.

The (\dashv) direction follows from C-UNASSOC:

$$C_{\mu \prec \kappa} (v', w'). K(v', w') \vdash C_\mu v. C_{\text{bind}(\kappa(v), \lambda w. \delta_{(v, w)})} (v', w'). K(v', w') \quad \text{(C-UNASSOC)}$$

$$\begin{aligned}
&\vdash C_\mu v. C_{\kappa(v)} w. C_{\delta(v,w)}(v', w'). K(v', w') && \text{(C-UNASSOC)} \\
&\vdash C_\mu v. C_{\kappa(v)} w. K(v, w) && \text{(C-UNIT-L)}
\end{aligned}$$

□

LEMMA F.41. *Section A is sound.*

PROOF.

$$\begin{aligned}
C_{\mu_1} v_1. C_{\mu_2} v_2. K(v_1, v_2) &\vdash C_{\mu_1 \otimes \mu_2}(v_1, v_2). K(v_1, v_2) && \text{(C-FUSE)} \\
&\vdash C_{\mu_2} v_2. C_{\mu_1} v_1. K(v_1, v_2) && \text{(C-FUSE)}
\end{aligned}$$

Where

$$\mu_1 \otimes \mu_2 = \mu_1 \prec (\lambda_-, \mu_2) = \mu_2 \prec (\lambda_-, \mu_1)$$

justifies the applications of **C-FUSE**.

□

LEMMA F.42. *Section A is sound.*

PROOF. By **SURE-STR-CONVEX** with $K = \text{True}$.

□

LEMMA F.43. *Section A is sound.*

PROOF.

$$\begin{aligned}
C_\mu v. E\langle i \rangle \sim \mu' &\vdash C_\mu v. C_{\mu'} w. [E\langle i \rangle = w] && \text{(C-UNIT-R)} \\
&\vdash C_{\mu'} w. C_\mu v. [E\langle i \rangle = w] && \text{(C-SWAP)} \\
&\vdash C_{\mu'} w. [E\langle i \rangle = w] && \text{(SURE-CONVEX)} \\
&\vdash E\langle i \rangle \sim \mu' && \text{(C-UNIT-R)}
\end{aligned}$$

□

LEMMA F.44. *The following rule is sound:*

$$\frac{\forall (v, _) \in \text{supp}(\mu). \forall \mu'. C_{\mu'} w. P(v) \vdash P(v)}{C_\mu(v, w). P(v) \dashv\vdash C_{\mu \circ \pi^{-1}} v. P(v)}$$

PROOF. Assume that for all $(v, _) \in \text{supp}(\mu)$, $\forall \mu'. C_{\mu'} w. P(v) \vdash P(v)$ (i.e. $P(v)$ is convex). By Lemma B.4 there is some κ such that $\mu = (\mu \circ \pi^{-1}) \prec \kappa$. Then:

$$\begin{aligned}
C_\mu(v, w). P(v) &\dashv\vdash C_{\mu \circ \pi^{-1}} v. C_{\kappa(v)} w. P(v) && \text{(C-FUSE)} \\
&\dashv\vdash C_{\mu \circ \pi^{-1}} v. P(v)
\end{aligned}$$

The last step is justified by the convexity assumption in the (\vdash) direction, and by **C-TRUE** and **C-FRAME** in the (\dashv) direction.

□

LEMMA F.45. *Section A is sound.*

PROOF. By Lemma F.44 and Lemma F.42.

□

LEMMA F.46. *Section A is sound.*

PROOF. Let $X_i \triangleq \{x \mid x\langle i \rangle \in X\}$ for every $i \in I$. Then:

$$\begin{aligned}
C_\mu(v, w). [x\langle i \rangle = v(x\langle i \rangle)]_{x\langle i \rangle \in X} &\dashv\vdash C_\mu(v, w). \bigwedge_{i \in I} [\bigwedge_{x \in X_i} x\langle i \rangle = v(x\langle i \rangle)] \\
&\dashv\vdash \bigwedge_{i \in I} C_\mu(v, w). [\bigwedge_{x \in X_i} x\langle i \rangle = v(x\langle i \rangle)] && \text{(C-AND)} \\
&\dashv\vdash \bigwedge_{i \in I} C_{\mu \circ \pi_i^{-1}} v. [\bigwedge_{x \in X_i} x\langle i \rangle = v(x\langle i \rangle)] && \text{(C-SURE-PROJ)}
\end{aligned}$$

$$\begin{aligned}
&\vdash C_{\mu \circ \pi_1^{-1}} \mathbf{v}. \bigwedge_{i \in I} \lceil \bigwedge_{x \in X_i} x \langle i \rangle = \mathbf{v}(x \langle i \rangle) \rceil & (\mathbf{C-AND}) \\
&\vdash C_{\mu \circ \pi_1^{-1}} \mathbf{v}. \lceil x \langle i \rangle = \mathbf{v}(x \langle i \rangle) \rceil_{x \langle i \rangle \in X}
\end{aligned}$$

Note that the (iterated) applications of **C-AND** satisfy the side condition because the inner assertions are by construction on disjoint indices. The backward direction of **C-AND** holds by the standard laws of conjunction. \square

LEMMA F.47. *Section 5.2 is sound.*

PROOF.

$$\begin{aligned}
C_{\mu_1} v_1. (\lceil E_1 \langle i \rangle = v_1 \rceil * E_2 \langle i \rangle \sim \mu_2) &\vdash C_{\mu_1} v_1. (\lceil E_1 \langle i \rangle = v_1 \rceil * C_{\mu_2} v_2. \lceil E_2 \langle i \rangle = v_2 \rceil) & (\mathbf{C-UNIT-R}) \\
&\vdash C_{\mu_1} v_1. C_{\mu_2} v_2. (\lceil E_1 \langle i \rangle = v_1 \rceil * \lceil E_2 \langle i \rangle = v_2 \rceil) & (\mathbf{C-FRAME}) \\
&\vdash C_{\mu_1} v_1. C_{\mu_2} v_2. \lceil E_1 \langle i \rangle = v_1 \wedge E_2 \langle i \rangle = v_2 \rceil & (\mathbf{SURE-MERGE}) \\
&\vdash C_{\mu_1 \otimes \mu_2} (v_1, v_2). \lceil (E_1 \langle i \rangle, E_2 \langle i \rangle) = (v_1, v_2) \rceil & (\mathbf{C-ASSOC}) \\
&\vdash (E_1 \langle i \rangle, E_2 \langle i \rangle) \sim (\mu_1 \otimes \mu_2) & (\mathbf{C-UNIT-R}) \\
&\vdash E_1 \langle i \rangle \sim \mu_1 * E_2 \langle i \rangle \sim \mu_2 & (\mathbf{PROD-SPLIT})
\end{aligned}$$

\square

LEMMA F.48. *Section A is sound.*

PROOF. By Lemma F.44 and Lemma F.43. \square

F.3.3 Relational Lifting.

LEMMA F.49. *Section A is sound.*

PROOF.

$$\begin{aligned}
\lfloor R_1 \rfloor &= \exists \mu. \lceil \mu(R_1) = 1^\top * C_\mu \mathbf{v}. \lceil x \langle i \rangle = \mathbf{v}(x \langle i \rangle) \rceil_{x \langle i \rangle \in X} \\
&\vdash \exists \mu. \lceil \mu(R_2) = 1^\top * C_\mu \mathbf{v}. \lceil x \langle i \rangle = \mathbf{v}(x \langle i \rangle) \rceil_{x \langle i \rangle \in X} & (\text{By } R_1 \subseteq R_2) \\
&= \lfloor R_2 \rfloor & \square
\end{aligned}$$

LEMMA F.50. *Section A is sound.*

PROOF.

$$\begin{aligned}
\lfloor R \rfloor &= \exists \mu. \lceil \mu(R) = 1^\top * C_\mu \mathbf{v}. \lceil x \langle i \rangle = \mathbf{v}(x \langle i \rangle) \rceil_{x \langle i \rangle \in X} \\
&\vdash \exists \mu. C_\mu \mathbf{v}. \lceil \mathbf{v} \in R^\top * \lceil x \langle i \rangle = \mathbf{v}(x \langle i \rangle) \rceil_{x \langle i \rangle \in X} & (\mathbf{C-PURE}) \\
&\vdash \exists \mu. C_\mu \mathbf{v}. \lceil R(x_1 \langle i \rangle, \dots, x_n \langle i \rangle) \rceil \\
&\vdash \exists \mu. \lceil R(x_1 \langle i \rangle, \dots, x_n \langle i \rangle) \rceil & (\mathbf{SURE-CONVEX}) \\
&\vdash \lceil R(x_1 \langle i \rangle, \dots, x_n \langle i \rangle) \rceil & \square
\end{aligned}$$

LEMMA F.51. *Section A is sound.*

PROOF.

$$\begin{aligned}
\lfloor x \langle i \rangle = y \langle j \rangle \rfloor &\vdash \exists \mu'. C_{\mu'} (v_1, v_2). (\lceil x \langle i \rangle = v_1 \rceil \wedge \lceil y \langle j \rangle = v_2 \rceil \wedge \lceil v_1 = v_2^\top \rceil) \\
&\vdash \exists \mu'. C_{\mu'} (v_1, v_2). (\lceil x \langle i \rangle = v_1 \rceil \wedge \lceil y \langle j \rangle = v_1 \rceil) & (\mathbf{C-CONS}) \\
&\vdash \exists \mu'. C_{\mu' \circ \pi_1^{-1}} v_1. (\lceil x \langle i \rangle = v_1 \rceil \wedge \lceil y \langle j \rangle = v_1 \rceil) & (\mathbf{C-SURE-PROJ}) \\
&\vdash \exists \mu. C_\mu v_1. (\lceil x \langle i \rangle = v_1 \rceil \wedge \lceil y \langle j \rangle = v_1 \rceil) & (\text{By } \mu = \mu' \circ \pi_1^{-1})
\end{aligned}$$

$$\begin{aligned}
& \vdash \exists \mu. (C_\mu v_1. \lceil x\langle i \rangle = v_1 \rceil) \wedge (C_\mu v_1. \lceil y\langle j \rangle = v_1 \rceil) \\
& \vdash \exists \mu. x\langle i \rangle \sim \mu \wedge y\langle j \rangle \sim \mu \quad (\text{C-UNIT-R}) \\
& \vdash \exists \mu. x\langle i \rangle \sim \mu * y\langle j \rangle \sim \mu \quad (\text{AND-TO-STAR})
\end{aligned}$$

□

LEMMA F.52. Section 2.3 is sound.

PROOF.

$$\begin{aligned}
C_\mu a. [R] &= C_\mu a. \exists \mu'. \lceil \mu'(R) = 1 \rceil * (C_{\mu'} v. \lceil x\langle i \rangle = v(x\langle i \rangle) \rceil_{x\langle i \rangle \in X}) \\
&\vdash \exists \kappa. C_\mu a. (C_{\kappa(a)} v. \lceil x\langle i \rangle = v(x\langle i \rangle) \rceil_{x\langle i \rangle \in X} * \lceil R(v) \rceil) \quad (\text{C-PURE, C-SKOLEM}) \\
&\vdash \exists \hat{\mu}. C_{\hat{\mu}}(a, v). \lceil x\langle i \rangle = v(x\langle i \rangle) \rceil_{x\langle i \rangle \in X} * \lceil R(v) \rceil \quad (\text{C-FUSE}) \\
&\vdash \exists \hat{\mu}. \lceil \hat{\mu} \circ \pi_2^{-1}(R) = 1 \rceil * C_{\hat{\mu}}(a, v). \lceil x\langle i \rangle = v(x\langle i \rangle) \rceil_{x\langle i \rangle \in X} \quad (\text{C-PURE}) \\
&\vdash \exists \hat{\mu}. \lceil \hat{\mu} \circ \pi_2^{-1}(R) = 1 \rceil * C_{\hat{\mu} \circ \pi_2^{-1}} v. \lceil x\langle i \rangle = v(x\langle i \rangle) \rceil_{x\langle i \rangle \in X} \quad (\text{C-SURE-PROJ-MANY}) \\
&\vdash \exists \hat{\mu}'. \lceil \hat{\mu}'(R) = 1 \rceil * C_{\hat{\mu}'} v. \lceil x\langle i \rangle = v(x\langle i \rangle) \rceil_{x\langle i \rangle \in X} \\
&= [R]
\end{aligned}$$

In the derivation we use $\hat{\mu} = \mu \prec \kappa$, and $\hat{\mu}' = \hat{\mu} \circ \pi_2^{-1}$. □

LEMMA F.53. Section 2.4 is sound.

PROOF. Let $R_1 \in \mathbb{V}^{X_1}$ and $R_2 \in \mathbb{V}^{X_2}$ and let $X = X_1 \cap X_2$, $Y_1 = X_1 \setminus X$, and $Y_2 = X_2 \setminus X$, so that $X_1 \cup X_2 = Y_1 \uplus X \uplus Y_2$.

By definition, $[R_1] * [R_2]$ entails that for some μ_1, μ_2 with $\mu_1(R_1) = 1$ and $\mu_1(R_2) = 1$:

$$\begin{aligned}
& C_{\mu_1} v_1. (\lceil x\langle i \rangle = v_1(x\langle i \rangle) \rceil_{x\langle i \rangle \in X_1} * C_{\mu_2} v_2. (\lceil x\langle i \rangle = v_2(x\langle i \rangle) \rceil_{x\langle i \rangle \in X_2})) \\
& \vdash C_{\mu_1}(\mathbf{w}_1, v_1). (\lceil y\langle i \rangle = \mathbf{w}_1(y\langle i \rangle) \rceil_{y\langle i \rangle \in Y_1} \wedge \lceil x\langle i \rangle = v_1(x\langle i \rangle) \rceil_{x\langle i \rangle \in X} * \lceil (\mathbf{w}_1 v_1) \in R_1 \rceil) * \\
& C_{\mu_2}(\mathbf{w}_2, v_2). (\lceil y\langle i \rangle = \mathbf{w}_2(y\langle i \rangle) \rceil_{y\langle i \rangle \in Y_2} \wedge \lceil x\langle i \rangle = v_2(x\langle i \rangle) \rceil_{x\langle i \rangle \in X} * \lceil (\mathbf{w}_2 v_2) \in R_2 \rceil) \quad (\text{C-PURE}) \\
& \vdash C_{\mu_1}(\mathbf{w}_1, v_1). C_{\mu_2}(\mathbf{w}_2, v_2). \left(\begin{array}{l} \lceil y\langle i \rangle = \mathbf{w}_1(y\langle i \rangle) \rceil_{y\langle i \rangle \in Y_1} \wedge \lceil x\langle i \rangle = v_1(x\langle i \rangle) \rceil_{x\langle i \rangle \in X} * \\ \lceil y\langle i \rangle = \mathbf{w}_2(y\langle i \rangle) \rceil_{y\langle i \rangle \in Y_2} \wedge \lceil x\langle i \rangle = v_2(x\langle i \rangle) \rceil_{x\langle i \rangle \in X} * \\ \lceil (\mathbf{w}_1 v_1) \in R_1 \rceil * \lceil (\mathbf{w}_2 v_2) \in R_2 \rceil \end{array} \right) \quad (\text{C-FRAME}) \\
& \vdash C_{\mu_1}(\mathbf{w}_1, v_1). C_{\mu_2}(\mathbf{w}_2, v_2). \left(\begin{array}{l} \lceil y\langle i \rangle = \mathbf{w}_1(y\langle i \rangle) \rceil_{y\langle i \rangle \in Y_1} \wedge \lceil x\langle i \rangle = v_1(x\langle i \rangle) \rceil_{x\langle i \rangle \in X} * \\ \lceil y\langle i \rangle = \mathbf{w}_2(y\langle i \rangle) \rceil_{y\langle i \rangle \in Y_2} \wedge \lceil x\langle i \rangle = v_2(x\langle i \rangle) \rceil_{x\langle i \rangle \in X} * \\ \lceil (\mathbf{w}_1 v_1) \in R_1 \rceil * \lceil (\mathbf{w}_2 v_2) \in R_2 \rceil * \lceil v_1 = v_2 \rceil \end{array} \right) \quad (\text{SURE-EQ-INJ}) \\
& \vdash C_{\mu_1}(\mathbf{w}_1, v_1). C_{\mu_2}(\mathbf{w}_2, v_2). \left(\begin{array}{l} \lceil y\langle i \rangle = \mathbf{w}_1(y\langle i \rangle) \rceil_{y\langle i \rangle \in Y_1} \wedge \\ \lceil x\langle i \rangle = v_1(x\langle i \rangle) \rceil_{x\langle i \rangle \in X} \wedge \\ \lceil y\langle i \rangle = \mathbf{w}_2(y\langle i \rangle) \rceil_{y\langle i \rangle \in Y_2} * \\ \lceil (\mathbf{w}_1 v_1) \in R_1 \wedge (\mathbf{w}_2 v_1) \in R_2 \rceil \end{array} \right) \quad (\text{C-CONS}) \\
& \vdash C_{\mu_1}(\mathbf{w}_1, v_1). C_{\mu_2 \circ \pi_1^{-1}}(\mathbf{w}_2). \left(\begin{array}{l} \lceil y\langle i \rangle = \mathbf{w}_1(y\langle i \rangle) \rceil_{y\langle i \rangle \in Y_1} \wedge \\ \lceil x\langle i \rangle = v_1(x\langle i \rangle) \rceil_{x\langle i \rangle \in X} \wedge \\ \lceil y\langle i \rangle = \mathbf{w}_2(y\langle i \rangle) \rceil_{y\langle i \rangle \in Y_2} * \\ \lceil (\mathbf{w}_1 v_1) \in R_1 \wedge (\mathbf{w}_2 v_1) \in R_2 \rceil \end{array} \right) \quad (\text{C-SURE-PROJ})
\end{aligned}$$

Thus by letting $\mu = \mu_1 \otimes (\mu_2 \circ \pi_1^{-1}) = \text{bind}(\mu_1, \kappa_2)$ where

$$\kappa_2 = \lambda(\mathbf{w}_1 v_1). (\text{bind}(\mu_2, \lambda(\mathbf{w}_2, v_2). \text{return}(\mathbf{w}_1 \mathbf{w}_2 v_1)))$$

we obtain:

$$\begin{aligned}
& C_{\mu_1}(\mathbf{w}'_1, \mathbf{v}'_1) \cdot C_{\kappa_2}(\mathbf{w}'_1, \mathbf{v}'_1)(\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}) \cdot \left(\begin{array}{l} \lceil y\langle i \rangle = \mathbf{w}_1(y\langle i \rangle) \rceil_{y\langle i \rangle \in Y_1} \wedge \\ \lceil x\langle i \rangle = \mathbf{w}(x\langle i \rangle) \rceil_{x\langle i \rangle \in X} \wedge \\ \lceil y\langle i \rangle = \mathbf{w}_2(y\langle i \rangle) \rceil_{y\langle i \rangle \in Y_2} * \\ \lceil (\mathbf{w}_1 \mathbf{w}) \in R_1 \wedge (\mathbf{w}_2 \mathbf{w}) \in R_2 \rceil \end{array} \right) \\
& \vdash C_{\mu}(\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}) \cdot \left(\begin{array}{l} \lceil y\langle i \rangle = \mathbf{w}_1(y\langle i \rangle) \rceil_{y\langle i \rangle \in Y_1} \wedge \\ \lceil x\langle i \rangle = \mathbf{w}(x\langle i \rangle) \rceil_{x\langle i \rangle \in X} \wedge \\ \lceil y\langle i \rangle = \mathbf{w}_2(y\langle i \rangle) \rceil_{y\langle i \rangle \in Y_2} \\ \lceil (\mathbf{w}_1 \mathbf{w}) \in R_1 \wedge (\mathbf{w}_2 \mathbf{w}) \in R_2 \rceil \end{array} \right) \quad (\text{C-ASSOC, C-SURE-PROJ}) \\
& \vdash C_{\mu} \mathbf{v} \cdot \lceil x\langle i \rangle = \mathbf{v}(x\langle i \rangle) \rceil_{x\langle i \rangle \in (X_1 \cup X_2)} * \lceil (\mathbf{v}|_{X_1}) \in R_1 \wedge (\mathbf{v}|_{X_2}) \in R_2 \rceil
\end{aligned}$$

The result gives us $\llbracket R_1 \wedge R_2 \rrbracket$ by **C-PURE** and Definition 4.12. \square

LEMMA F.54. Section A is sound.

PROOF.

$$\begin{aligned}
& \llbracket R \rrbracket * \lceil x\langle i \rangle = e\langle i \rangle \rceil \vdash \exists \mu. C_{\mu} \mathbf{v} \cdot (\lceil y\langle i \rangle = \mathbf{v}(y\langle i \rangle) \rceil_{y\langle i \rangle \in X} * \lceil R(\mathbf{v}) \rceil) * \lceil x\langle i \rangle = e\langle i \rangle \rceil \quad (\text{By def.}) \\
& \vdash \exists \mu. C_{\mu} \mathbf{v} \cdot (\lceil y\langle i \rangle = \mathbf{v}(y\langle i \rangle) \rceil_{y\langle i \rangle \in X} * \lceil R(\mathbf{v}) \rceil * \lceil x\langle i \rangle = e\langle i \rangle \rceil) \quad (\text{C-FRAME}) \\
& \vdash \exists \mu. C_{\mu} \mathbf{v} \cdot (\lceil y\langle i \rangle = \mathbf{v}(y\langle i \rangle) \rceil_{y\langle i \rangle \in X} * \lceil R(\mathbf{v}) \rceil * \lceil x\langle i \rangle = \llbracket e\langle i \rangle \rrbracket(\mathbf{v}) \rceil) \\
& \quad (\text{By pvar}(e\langle i \rangle) \subseteq X) \\
& \vdash \exists \mu. C_{\mu} \mathbf{v} \cdot (\lceil y\langle i \rangle = \mathbf{v}(y\langle i \rangle) \rceil_{y\langle i \rangle \in X} * \lceil R(\mathbf{v}) \rceil * C_{\delta_{\llbracket e\langle i \rangle \rrbracket(\mathbf{v})}} \mathbf{w} \cdot \lceil x\langle i \rangle = \mathbf{w} \rceil) \\
& \quad (\text{C-UNIT-L}) \\
& \vdash \exists \mu. C_{\mu} \mathbf{v} \cdot C_{\delta_{\llbracket e\langle i \rangle \rrbracket(\mathbf{v})}} \mathbf{w} \cdot (\lceil y\langle i \rangle = \mathbf{v}(y\langle i \rangle) \rceil_{y\langle i \rangle \in X} * \lceil R(\mathbf{v}) \rceil * \lceil x\langle i \rangle = \mathbf{w} \rceil) \\
& \quad (\text{C-FRAME}) \\
& \vdash \exists \mu'. C_{\mu'} \mathbf{v}' \cdot \left(\begin{array}{l} \lceil y\langle i \rangle = \mathbf{v}'(y\langle i \rangle) \rceil_{y\langle i \rangle \in X} * \lceil x\langle i \rangle = \mathbf{v}'(x\langle i \rangle) \rceil \\ * \lceil R(\mathbf{v}') \wedge \llbracket e\langle i \rangle \rrbracket(\mathbf{v}') = \mathbf{v}'(x\langle i \rangle) \rceil \end{array} \right) \quad (\text{C-PURE, C-ASSOC}) \\
& \vdash \llbracket R \wedge x\langle i \rangle = e\langle i \rangle \rrbracket
\end{aligned}$$

where we let $\mu' \triangleq (\mathbf{v} \leftarrow \mu; \text{return}(\mathbf{v}[x\langle i \rangle] \mapsto \llbracket e\langle i \rangle \rrbracket(\mathbf{v})))$. \square

LEMMA F.55. Section 2.3 is sound.

PROOF. Assuming $\mu \circ \pi_1^{-1} = \mu_1$, $\mu \circ \pi_2^{-1} = \mu_2$, and $\mu(R) = 1$, we have:

$$\begin{aligned}
& x_1\langle 1 \rangle \sim \mu_1 * x_2\langle 2 \rangle \sim \mu_2 \vdash C_{\mu_1} \mathbf{v} \cdot \lceil x_1\langle 1 \rangle = \mathbf{v} \rceil * C_{\mu_2} \mathbf{w} \cdot \lceil x_2\langle 2 \rangle = \mathbf{w} \rceil \quad (\text{C-UNIT-R}) \\
& \vdash C_{\mu}(\mathbf{v}, \mathbf{w}) \cdot \lceil x_1\langle 1 \rangle = \mathbf{v} \rceil * C_{\mu}(\mathbf{v}, \mathbf{w}) \cdot \lceil x_2\langle 2 \rangle = \mathbf{w} \rceil \quad (\text{C-SURE-PROJ}) \\
& \vdash C_{\mu}(\mathbf{v}, \mathbf{w}) \cdot \lceil x_1\langle 1 \rangle = \mathbf{v} \rceil \wedge C_{\mu}(\mathbf{v}, \mathbf{w}) \cdot \lceil x_2\langle 2 \rangle = \mathbf{w} \rceil \quad (\text{AND-TO-STAR}) \\
& \vdash C_{\mu}(\mathbf{v}, \mathbf{w}) \cdot (\lceil x_1\langle 1 \rangle = \mathbf{v} \rceil \wedge \lceil x_2\langle 2 \rangle = \mathbf{w} \rceil) \quad (\text{C-AND}) \\
& \vdash \llbracket R(x_1\langle 1 \rangle, x_2\langle 2 \rangle) \rrbracket \quad (\text{By } \mu(R) = 1)
\end{aligned}$$

\square

F.3.4 Weakest Precondition.

LEMMA F.56. Section A is sound.

PROOF. Special case of **WP-LOOP** with $n = 0$, which makes the premises trivial. \square

LEMMA F.57. Section A is sound.

PROOF. We derive the following rule:

$$\frac{\forall k < n. P(k) \vdash \mathbf{wp} [i: t, j: t'] \{P(k+1)\}}{P(0) \vdash \mathbf{wp} [i: (\mathbf{repeat} \ n \ t), j: (\mathbf{repeat} \ n \ t')] \{P(n)\}}$$

(for $n \in \mathbb{N}$ and $i \neq j$) from the standard **WP-LOOP**, as follows. Let

$$P'(k) \triangleq \mathbf{wp} [j: \mathbf{repeat} \ k \ t'] \{P(k)\}$$

Note that $P(0) \vdash P'(0)$ by **WP-LOOP-0**. Then we can apply the **WP-LOOP** using P' as a loop invariant

$$\begin{array}{c} \frac{\forall k \leq n. P(k) \vdash \mathbf{wp} [i: t, j: t'] \{P(k+1)\}}{\forall k \leq n. P(k) \vdash \mathbf{wp} [j: t'] \{\mathbf{wp} [i: t] \{P(k+1)\}\}} \text{WP-NEST} \\ \frac{\forall k \leq n. \mathbf{wp} [j: \mathbf{repeat} \ k \ t'] \{P(k)\} \vdash \mathbf{wp} [j: \mathbf{repeat} \ k \ t'] \{\mathbf{wp} [i: t] \{\mathbf{wp} [j: t'] \{\mathbf{wp} [i: t] \{P(k+1)\}\}\}\}}{\forall k \leq n. \mathbf{wp} [j: \mathbf{repeat} \ k \ t'] \{P(k)\} \vdash \mathbf{wp} [j: \mathbf{repeat} \ (k+1) \ t'] \{\mathbf{wp} [i: t] \{P(k+1)\}\}} \text{WP-CONS} \\ \frac{\forall k \leq n. \mathbf{wp} [j: \mathbf{repeat} \ k \ t'] \{P(k)\} \vdash \mathbf{wp} [j: \mathbf{repeat} \ (k+1) \ t'] \{\mathbf{wp} [i: t] \{P(k+1)\}\}}{\forall k \leq n. \mathbf{wp} [j: \mathbf{repeat} \ k \ t'] \{P(k)\} \vdash \mathbf{wp} [i: t] \{\mathbf{wp} [j: \mathbf{repeat} \ (k+1) \ t'] \{P(k+1)\}\}} \text{WP-LOOP-UNF} \\ \frac{\forall k \leq n. P(k)' \vdash \mathbf{wp} [i: t] \{P'(k+1)\}}{P'(0) \vdash \mathbf{wp} [i: (\mathbf{repeat} \ n \ t)] \{P'(n)\}} \text{WP-NEST} \\ \frac{P'(0) \vdash \mathbf{wp} [i: (\mathbf{repeat} \ n \ t)] \{P'(n)\}}{P(0) \vdash \mathbf{wp} [i: (\mathbf{repeat} \ n \ t)] \{\mathbf{wp} [j: (\mathbf{repeat} \ n \ t')] \{P(n)\}\}} \text{WP-LOOP} \\ \frac{P(0) \vdash \mathbf{wp} [i: (\mathbf{repeat} \ n \ t)] \{\mathbf{wp} [j: (\mathbf{repeat} \ n \ t')] \{P(n)\}\}}{P(0) \vdash \mathbf{wp} [i: (\mathbf{repeat} \ n \ t), j: (\mathbf{repeat} \ n \ t')] \{P(n)\}} \text{WP-NEST} \end{array}$$

From bottom to top, we focus on component i using **WP-NEST**; then we use $P(0) \vdash P'(0)$ and transitivity of entailment to rewrite the goal using the invariant P' ; we then use **WP-LOOP** and unfold the invariant; using **WP-NEST** twice we can swap the two components so that component j is the topmost WP in the assumption and conclusion; using **WP-LOOP-UNF** we break off the first k iterations at j ; finally, using **WP-CONS** we can eliminate the topmost WP on both sides of the entailments.

It is straightforward to adapt the argument for any number of components looping the same number of times. \square

LEMMA F.58. *Section A is sound.*

PROOF. Define $\mathbf{p}_R \triangleq (\mathbf{p} \setminus \mathbf{x}\langle i \rangle)/2$ and $\mathbf{p}_x \triangleq \mathbf{p} - \mathbf{p}_R$; note that by $\mathbf{p}(\mathbf{x}\langle i \rangle) = 1$ we have $\mathbf{p}_x(\mathbf{x}\langle i \rangle) = 1$. We first show that the following hold:

$$\llbracket R \rrbracket @ \mathbf{p} \vdash \llbracket R \rrbracket @ \mathbf{p}_R * (\mathbf{p}_x) \quad (71)$$

$$\llbracket R \rrbracket @ \mathbf{p}_R * \llbracket \mathbf{x}\langle i \rangle = e \langle i \rangle \rrbracket @ \mathbf{p}_x \vdash \llbracket R \wedge \mathbf{x}\langle i \rangle = e \langle i \rangle \rrbracket @ \mathbf{p} \quad (72)$$

The first entailment holds because $\llbracket R \rrbracket$ is permission-scaling-invariant (see Section B.2) and by the assumption that $\mathbf{x} \notin \text{pvar}(R)$. The second entailment holds by **RL-SURE-MERGE**.

We can then derive:

$$\begin{array}{c} \frac{(\mathbf{p}_x) \vdash \mathbf{wp} [i: \mathbf{x} := e] \{\llbracket \mathbf{x}\langle i \rangle = e \langle i \rangle \rrbracket @ \mathbf{p}_x\}}{\llbracket R \rrbracket @ \mathbf{p}_R * (\mathbf{p}_x) \vdash \mathbf{wp} [i: \mathbf{x} := e] \{\llbracket R \rrbracket @ \mathbf{p}_R * \llbracket \mathbf{x}\langle i \rangle = e \langle i \rangle \rrbracket @ \mathbf{p}_x\}} \text{WP-ASSIGN} \\ \frac{\llbracket R \rrbracket @ \mathbf{p}_R * (\mathbf{p}_x) \vdash \mathbf{wp} [i: \mathbf{x} := e] \{\llbracket R \rrbracket @ \mathbf{p}_R * \llbracket \mathbf{x}\langle i \rangle = e \langle i \rangle \rrbracket @ \mathbf{p}_x\}}{\llbracket R \rrbracket @ \mathbf{p} \vdash \mathbf{wp} [i: \mathbf{x} := e] \{\llbracket R \wedge \mathbf{x}\langle i \rangle = e \langle i \rangle \rrbracket @ \mathbf{p}\}} \text{WP-FRAME} \\ \text{WP-CONS} \end{array} \quad \square$$

LEMMA F.59. *Section A is sound.*

PROOF. From the premises, we derive:

$$\begin{array}{c}
\frac{P * \lceil e\langle 1 \rangle = 1 \rceil \Vdash \mathbf{wp} [1: t_1] \{Q(1)\} \quad P * \lceil e\langle 1 \rangle = 0 \rceil \Vdash \mathbf{wp} [1: t_2] \{Q(0)\}}{\forall b \in \{0, 1\}. P * \lceil e\langle 1 \rangle = b \rceil \Vdash \text{if } b \text{ then } \mathbf{wp} [1: t_1] \{Q(1)\} \text{ else } \mathbf{wp} [1: t_2] \{Q(0)\}} \text{WP-IF-PRIM} \\
\frac{\forall b \in \{0, 1\}. P * \lceil e\langle 1 \rangle = b \rceil \Vdash \mathbf{wp} [1: (\text{if } b \text{ then } t_1 \text{ else } t_2)] \{Q(b \doteq 1)\}}{\forall b \in \{0, 1\}. P * \lceil e\langle 1 \rangle = b \rceil \Vdash \mathbf{wp} [1: (\text{if } e \text{ then } t_1 \text{ else } t_2)] \{Q(b \doteq 1)\}} \text{WP-BIND} \\
\frac{\forall b \in \{0, 1\}. P * \lceil e\langle 1 \rangle = b \rceil \Vdash \mathbf{wp} [1: (\text{if } e \text{ then } t_1 \text{ else } t_2)] \{Q(b \doteq 1)\}}{C_\beta b. (P * \lceil e\langle 1 \rangle = b \rceil) \Vdash C_\beta b. \mathbf{wp} [1: (\text{if } e \text{ then } t_1 \text{ else } t_2)] \{Q(b \doteq 1)\}} \text{C-CONS} \\
\frac{C_\beta b. (P * \lceil e\langle 1 \rangle = b \rceil) \Vdash C_\beta b. \mathbf{wp} [1: (\text{if } e \text{ then } t_1 \text{ else } t_2)] \{Q(b \doteq 1)\}}{P * e\langle 1 \rangle \sim \beta \Vdash C_\beta b. \mathbf{wp} [1: (\text{if } e \text{ then } t_1 \text{ else } t_2)] \{Q(b \doteq 1)\}} \text{C-UNIT-R, C-FRAME} \\
\frac{P * e\langle 1 \rangle \sim \beta \Vdash C_\beta b. \mathbf{wp} [1: (\text{if } e \text{ then } t_1 \text{ else } t_2)] \{Q(b \doteq 1)\}}{P * e\langle 1 \rangle \sim \beta \Vdash \mathbf{wp} [1: (\text{if } e \text{ then } t_1 \text{ else } t_2)] \{C_\beta b. Q(b \doteq 1)\}} \text{C-WP-SWAP} \quad \square
\end{array}$$

G Case Studies

G.1 pRHL-style Reasoning

Here we elaborate on the conditional swap example that appeared in Section 5.1. By Section 4.4, for each index $i \in \{1, 2\}$, we have

$$\Vdash \mathbf{wp} [i: x \approx d_0] \{x\langle i \rangle \sim d_0\}$$

By Section A, we can combine the two programs together and derive

$$\Vdash \mathbf{wp} [1: x \approx d_0, 2: x \approx d_0] \{x\langle 1 \rangle \sim d_0 \wedge x\langle 2 \rangle \sim d_0\}$$

By Section 2.3,

$$x\langle 1 \rangle \sim d_0 \wedge x\langle 2 \rangle \sim d_0 \vdash C_{d_0} v. \lceil x\langle 1 \rangle = v \rceil \wedge C_{d_0} v. \lceil x\langle 2 \rangle = v \rceil$$

Then, we can apply Section 4.3, which implies

$$C_{d_0} v. \lceil x\langle 1 \rangle = v \rceil \wedge C_{d_0} v. \lceil x\langle 2 \rangle = v \rceil \vdash C_{d_0} v. (\lceil x\langle 1 \rangle = v \rceil \wedge \lceil x\langle 2 \rangle = v \rceil)$$

from which we can derive:

$$\Vdash \mathbf{wp} [1: x \approx d_0, 2: x \approx d_0] \{C_{d_0} v. (\lceil x\langle 1 \rangle = v \rceil \wedge \lceil x\langle 2 \rangle = v \rceil)\}.$$

For the rest of prog1 (prog2): similarly, by Section 4.4, for each index $i \in \{1, 2\}$, we have

$$\Vdash \mathbf{wp} [i: y \approx d_1(v)] \{y\langle i \rangle \sim d_1(v)\} \quad (73)$$

$$\Vdash \mathbf{wp} [i: z \approx d_2(v)] \{z\langle i \rangle \sim d_2(v)\}. \quad (74)$$

By Section 2.4,

$$z\langle i \rangle \sim d_2(v) \Vdash \mathbf{wp} [i: y \approx d_1(v)] \{z\langle i \rangle \sim d_2(v) * y\langle i \rangle \sim d_1(v)\} \quad (75)$$

$$y\langle i \rangle \sim d_1(v) \Vdash \mathbf{wp} [i: z \approx d_2(v)] \{y\langle i \rangle \sim d_1(v) * z\langle i \rangle \sim d_2(v)\}. \quad (76)$$

Thus, applying Section 2.4 to combine Eq. (73) and Eq. (76), we get

$$\Vdash \mathbf{wp} [1: y \approx d_1(v); z \approx d_2(v)] \{y\langle 1 \rangle \sim d_1(v) * z\langle 1 \rangle \sim d_2(v)\}; \quad (77)$$

By applying Section 2.4 to combine Eq. (74) and Eq. (75), we get

$$\Vdash \mathbf{wp} [2: z \approx d_2(v); y \approx d_1(v)] \{y\langle 2 \rangle \sim d_1(v) * z\langle 2 \rangle \sim d_2(v)\}. \quad (78)$$

Then, by Section A, we can derive

$$\begin{aligned}
&\lceil x\langle 1 \rangle = v \rceil \Vdash \mathbf{wp} [1: y \approx d_1(x); z \approx d_2(x)] \\
&\quad \hookrightarrow \{y\langle 1 \rangle \sim d_1(v) * z\langle 1 \rangle \sim d_2(v)\} \\
&\lceil x\langle 2 \rangle = v \rceil \Vdash \mathbf{wp} [2: z \approx d_2(v); y \approx d_1(v)] \{y\langle 2 \rangle \sim d_1(v) * z\langle 2 \rangle \sim d_2(v)\}
\end{aligned}$$

Then, applying Section A to combine the program at index 1 and 2, we get

$$\begin{aligned} \lceil x\langle 1 \rangle = v \rceil \wedge \lceil x\langle 2 \rangle = v \rceil \Vdash \mathbf{wp} \left[\begin{array}{l} 1: y \approx d_1(x); z \approx d_2(x); 2: z \approx d_2(v); y \approx d_1(v) \\ \hookrightarrow \{y\langle 1 \rangle \sim d_1(v) * z\langle 1 \rangle \sim d_2(v) * y\langle 2 \rangle \sim d_1(v) * z\langle 2 \rangle \sim d_2(v)\} \end{array} \right] \end{aligned}$$

Also, we have

$$\begin{aligned} \lceil x\langle 1 \rangle = v \rceil \wedge \lceil x\langle 2 \rangle = v \rceil \Vdash \mathbf{wp} \left[\begin{array}{l} 1: y \approx d_1(x); z \approx d_2(x); 2: z \approx d_2(v); y \approx d_1(v) \\ \hookrightarrow \{ \lceil x\langle 1 \rangle = v \rceil \wedge \lceil x\langle 2 \rangle = v \rceil \} \end{array} \right] \end{aligned}$$

by Section 2.4, where $\lceil x\langle 1 \rangle = v \rceil \wedge \lceil x\langle 2 \rangle = v \rceil \vdash \lceil x\langle 1 \rangle = x\langle 2 \rangle \rceil$. Therefore, we have

$$\begin{aligned} \lceil x\langle 1 \rangle = v \rceil \wedge \lceil x\langle 2 \rangle = v \rceil \Vdash \mathbf{wp} \left[\begin{array}{l} 1: y \approx d_1(x); z \approx d_2(x); 2: z \approx d_2(v); y \approx d_1(v) \\ \hookrightarrow \{ \lceil x\langle 1 \rangle = x\langle 2 \rangle \rceil \} \end{array} \right] \end{aligned}$$

By Section A,

$$\begin{aligned} \lceil x\langle 1 \rangle = v \rceil \wedge \lceil x\langle 2 \rangle = v \rceil \\ \Vdash \mathbf{wp} \left[\begin{array}{l} 1: y \approx d_1(x); z \approx d_2(x); 2: z \approx d_2(v); y \approx d_1(v) \\ \hookrightarrow \{ \lceil x\langle 1 \rangle = x\langle 2 \rangle \rceil \wedge (y\langle 1 \rangle \sim d_1(v) * z\langle 1 \rangle \sim d_2(v) * y\langle 2 \rangle \sim d_1(v) * z\langle 2 \rangle \sim d_2(v)) \} \end{array} \right] \end{aligned}$$

By Section A, we get

$$\begin{aligned} \lceil x\langle 1 \rangle = v \rceil \wedge \lceil x\langle 2 \rangle = v \rceil \\ \Vdash \mathbf{wp} \left[\begin{array}{l} 1: y \approx d_1(x); z \approx d_2(x); 2: z \approx d_2(v); y \approx d_1(v) \\ \hookrightarrow \{ \lceil x\langle 1 \rangle = x\langle 2 \rangle \rceil * y\langle 1 \rangle \sim d_1(v) * z\langle 1 \rangle \sim d_2(v) * y\langle 2 \rangle \sim d_1(v) * z\langle 2 \rangle \sim d_2(v) \} \end{array} \right] \end{aligned}$$

Now, we can proceed with the derivation explained in Section 5.1.

$$\begin{array}{c} \forall v. \lceil x\langle 1 \rangle = v \rceil \wedge \lceil x\langle 2 \rangle = v \rceil \Vdash \mathbf{wp} \left[\begin{array}{l} 1: t_1, 2: t_2 \\ \left\{ \begin{array}{l} \lfloor x\langle 1 \rangle = x\langle 2 \rangle \rfloor * y\langle 1 \rangle \sim d_1(v) * y\langle 2 \rangle \sim d_1(v) * \\ \lfloor z\langle 1 \rangle \sim d_2(v) * z\langle 2 \rangle \sim d_2(v) \end{array} \right\} \end{array} \right] \\ \hline \forall v. \lceil x\langle 1 \rangle = v \rceil \wedge \lceil x\langle 2 \rangle = v \rceil \Vdash \mathbf{wp} \left[\begin{array}{l} 1: t_1, 2: t_2 \\ \left\{ \lfloor x\langle 1 \rangle = x\langle 2 \rangle \rfloor * \lfloor y\langle 1 \rangle = y\langle 2 \rangle \rfloor * \lfloor z\langle 1 \rangle = z\langle 2 \rangle \rfloor \right\} \end{array} \right] \quad \text{COUPLING} \\ \hline \forall v. \lceil x\langle 1 \rangle = v \rceil \wedge \lceil x\langle 2 \rangle = v \rceil \Vdash \mathbf{wp} \left[\begin{array}{l} 1: t_1, 2: t_2 \\ \left\{ \lfloor x\langle 1 \rangle = x\langle 2 \rangle \rfloor \wedge \lfloor y\langle 1 \rangle = y\langle 2 \rangle \rfloor \wedge \lfloor z\langle 1 \rangle = z\langle 2 \rangle \rfloor \right\} \end{array} \right] \quad \text{RL-MERGE} \\ \hline C_{d_0} v. (\lceil x\langle 1 \rangle = v \rceil \wedge \lceil x\langle 2 \rangle = v \rceil) \Vdash C_{d_0} v. \mathbf{wp} \left[\begin{array}{l} 1: t_1, 2: t_2 \\ \left\{ \lfloor x\langle 1 \rangle = x\langle 2 \rangle \rfloor \wedge \lfloor y\langle 1 \rangle = y\langle 2 \rangle \rfloor \wedge \lfloor z\langle 1 \rangle = z\langle 2 \rangle \rfloor \right\} \end{array} \right] \quad \text{C-CONS} \\ \hline C_{d_0} v. (\lceil x\langle 1 \rangle = v \rceil \wedge \lceil x\langle 2 \rangle = v \rceil) \Vdash \mathbf{wp} \left[\begin{array}{l} 1: t_1, 2: t_2 \\ \left\{ C_{d_0} v. \lfloor x\langle 1 \rangle = x\langle 2 \rangle \rfloor \wedge \lfloor y\langle 1 \rangle = y\langle 2 \rangle \rfloor \wedge \lfloor z\langle 1 \rangle = z\langle 2 \rangle \rfloor \right\} \end{array} \right] \quad \text{C-WP-SWAP} \\ \hline C_{d_0} v. (\lceil x\langle 1 \rangle = v \rceil \wedge \lceil x\langle 2 \rangle = v \rceil) \Vdash \mathbf{wp} \left[\begin{array}{l} 1: t_1, 2: t_2 \\ \left\{ \lfloor x\langle 1 \rangle = x\langle 2 \rangle \rfloor \wedge \lfloor y\langle 1 \rangle = y\langle 2 \rangle \rfloor \wedge \lfloor z\langle 1 \rangle = z\langle 2 \rangle \rfloor \right\} \end{array} \right] \quad \text{RL-CONVEX} \end{array}$$

Last, with Section 2.4, we have

$$\Vdash \mathbf{wp} \left[\begin{array}{l} 1: \text{prog1}, 2: \text{prog2} \\ \left\{ \lfloor x\langle 1 \rangle = x\langle 2 \rangle \rfloor \wedge \lfloor y\langle 1 \rangle = y\langle 2 \rangle \rfloor \wedge \lfloor z\langle 1 \rangle = z\langle 2 \rangle \rfloor \right\} \end{array} \right]$$

G.2 One-time Pad (Relational)

To wrap up the proof of Section 2 we first observe that the assertion P of (3) can be easily obtained by using the WP rules for assignments and sequencing, proving:

$$\text{True}@p \vdash \mathbf{wp} \left[\begin{array}{l} 1: \text{encrypt}() \\ 2: c \approx \text{Ber}(\frac{1}{2}) \end{array} \right] \left\{ \left(\begin{array}{l} k\langle 1 \rangle \sim \text{Ber}_{\frac{1}{2}} * m\langle 1 \rangle \sim \text{Ber}_p * c\langle 2 \rangle \sim \text{Ber}_{\frac{1}{2}} * \\ \lfloor c\langle 1 \rangle = k\langle 1 \rangle \text{ xor } m\langle 1 \rangle \rfloor \end{array} \right) @p \right\}$$

where $p = [k\langle 1 \rangle: 1, m\langle 1 \rangle: 1, c\langle 1 \rangle: 1, c\langle 2 \rangle: 1]$ (i.e. we have full permissions on the variables we modify).

We can prove the entailment:

$$C_{\text{Ber}_p} v. \left(\lceil m\langle 1 \rangle = v \rceil * \left(\begin{array}{l} k\langle 1 \rangle \sim \text{Ber}_{\frac{1}{2}} \\ * c\langle 2 \rangle \sim \text{Ber}_{\frac{1}{2}} \end{array} \right) \right) \vdash C_{\text{Ber}_p} v. \left(\lceil m\langle 1 \rangle = v \rceil * \left\{ \begin{array}{ll} \lfloor k\langle 1 \rangle = c\langle 2 \rangle \rfloor & \text{if } v = 0 \\ \lfloor k\langle 1 \rangle = \neg c\langle 2 \rangle \rfloor & \text{if } v = 1 \end{array} \right. \right)$$

by using C-CONS, which asks us to prove that the two assertions inside the conditioning are in the entailment relation for each value of v . This leads to these two cases:

$$\lceil m\langle 1 \rangle = 0 \rceil * k\langle 1 \rangle \sim \text{Ber}_{\frac{1}{2}} * c\langle 2 \rangle \sim \text{Ber}_{\frac{1}{2}} \vdash \lceil m\langle 1 \rangle = 0 \rceil * \lfloor k\langle 1 \rangle = c\langle 2 \rangle \rfloor$$

$$\lceil m\langle 1 \rangle = 1 \rceil * k\langle 1 \rangle \sim \text{Ber}_{\frac{1}{2}} * c\langle 2 \rangle \sim \text{Ber}_{\frac{1}{2}} \vdash \lceil m\langle 1 \rangle = 1 \rceil * \lfloor k\langle 1 \rangle = \neg c\langle 2 \rangle \rfloor$$

which are straightforward consequences of the two couplings we proved in (6).

Finally, the assignment to c in `encrypt` generated the fact $\lceil c\langle 1 \rangle = k\langle 1 \rangle \text{ xor } m\langle 1 \rangle \rceil$. By routine propagation of this fact (using **C-FRAME** and **SURE-MERGE**) we can establish:

$$\begin{aligned} & C_{\text{Ber}_p} v. \left(\lceil m\langle 1 \rangle = v \rceil * \begin{cases} \lfloor k\langle 1 \rangle = c\langle 2 \rangle \rfloor & \text{if } v = 0 \\ \lfloor k\langle 1 \rangle = \neg c\langle 2 \rangle \rfloor & \text{if } v = 1 \end{cases} * \lceil c\langle 1 \rangle = k\langle 1 \rangle \text{ xor } m\langle 1 \rangle \rceil \right) \\ \vdash & C_{\text{Ber}_p} v. \left(\lceil m\langle 1 \rangle = v \rceil * \begin{cases} \lfloor k\langle 1 \rangle = c\langle 2 \rangle \rfloor * \lceil c\langle 1 \rangle = k\langle 1 \rangle \text{ xor } 0 \rceil & \text{if } v = 0 \\ \lfloor k\langle 1 \rangle = \neg c\langle 2 \rangle \rfloor * \lceil c\langle 1 \rangle = k\langle 1 \rangle \text{ xor } 1 \rceil & \text{if } v = 1 \end{cases} \right) \\ \vdash & C_{\text{Ber}_p} v. \left(\lceil m\langle 1 \rangle = v \rceil * \begin{cases} \lfloor c\langle 1 \rangle = c\langle 2 \rangle \rfloor & \text{if } v = 0 \\ \lfloor c\langle 1 \rangle = c\langle 2 \rangle \rfloor & \text{if } v = 1 \end{cases} \right) \\ \vdash & C_{\text{Ber}_p} v. \lfloor c\langle 1 \rangle = c\langle 2 \rangle \rfloor \\ \vdash & \lfloor c\langle 1 \rangle = c\langle 2 \rangle \rfloor \quad (\text{RL-MERGE}) \end{aligned}$$

In particular, the entailments

$$\begin{aligned} \lfloor k\langle 1 \rangle = c\langle 2 \rangle \rfloor * \lceil c\langle 1 \rangle = k\langle 1 \rangle \text{ xor } 0 \rceil &\vdash \lfloor c\langle 1 \rangle = c\langle 2 \rangle \rfloor \\ \lfloor k\langle 1 \rangle = \neg c\langle 2 \rangle \rfloor * \lceil c\langle 1 \rangle = k\langle 1 \rangle \text{ xor } 1 \rceil &\vdash \lfloor c\langle 1 \rangle = c\langle 2 \rangle \rfloor \end{aligned}$$

can be proved by applying **RL-SURE-MERGE** and **RL-CONS**.

G.3 One-time Pad (Unary)

Similarly to the relational version, we can, using **WP-SEQ**, **WP-ASSIGN** and **WP-SAMP**, easily show that:

$$\text{True}@p \vdash \text{wp} [1: \text{encrypt}()] \{k\langle 1 \rangle \sim \text{Ber}_{\frac{1}{2}} * m\langle 1 \rangle \sim \text{Ber}_p * \lceil c\langle 1 \rangle = k\langle 1 \rangle \text{ xor } m\langle 1 \rangle \rceil\}.$$

We then show the crucial derivation of Section 5.2 in more detail.

$$\begin{aligned} & k\langle 1 \rangle \sim \text{Ber}_{\frac{1}{2}} * m\langle 1 \rangle \sim \text{Ber}_p * \lceil c\langle 1 \rangle = k\langle 1 \rangle \text{ xor } m\langle 1 \rangle \rceil \\ \vdash & C_{\text{Ber}_p} m. (\lceil m\langle 1 \rangle = m \rceil * k\langle 1 \rangle \sim \text{Ber}_{\frac{1}{2}} * \lceil c\langle 1 \rangle = k\langle 1 \rangle \text{ xor } m\langle 1 \rangle \rceil) \quad (\text{C-UNIT-R, C-FRAME}) \\ \vdash & C_{\text{Ber}_p} m. (\lceil m\langle 1 \rangle = m \rceil * k\langle 1 \rangle \sim \text{Ber}_{\frac{1}{2}} * \lceil c\langle 1 \rangle = k\langle 1 \rangle \text{ xor } m \rceil) \quad (\text{SURE-MERGE}) \\ \vdash & C_{\text{Ber}_p} m. \left(\lceil m\langle 1 \rangle = m \rceil * C_{\text{Ber}_{\frac{1}{2}}} k. (\lceil k\langle 1 \rangle = k \rceil * \lceil c\langle 1 \rangle = k\langle 1 \rangle \text{ xor } m \rceil) \right) \quad (\text{C-UNIT-R, C-FRAME}) \\ \vdash & C_{\text{Ber}_p} m. (\lceil m\langle 1 \rangle = m \rceil * C_{\text{Ber}_{\frac{1}{2}}} k. \lceil k\langle 1 \rangle = k \wedge c\langle 1 \rangle = k \text{ xor } m \rceil) \quad (\text{SURE-MERGE}) \\ \vdash & C_{\text{Ber}_p} m. \left(\lceil m\langle 1 \rangle = m \rceil * \begin{cases} C_{\text{Ber}_{\frac{1}{2}}} k. \lceil c\langle 1 \rangle = k \rceil & \text{if } m = 0 \\ C_{\text{Ber}_{\frac{1}{2}}} k. \lceil c\langle 1 \rangle = \neg k \rceil & \text{if } m = 1 \end{cases} \right) \quad (\text{C-CONS}) \\ \vdash & C_{\text{Ber}_p} m. \left(\lceil m\langle 1 \rangle = m \rceil * \begin{cases} C_{\text{Ber}_{\frac{1}{2}}} k. \lceil c\langle 1 \rangle = k \rceil & \text{if } m = 0 \\ C_{\text{Ber}_{\frac{1}{2}}} k. \lceil c\langle 1 \rangle = k \rceil & \text{if } m = 1 \end{cases} \right) \quad (\text{C-TRANSF}) \\ \vdash & C_{\text{Ber}_p} m. (\lceil m\langle 1 \rangle = m \rceil * C_{\text{Ber}_{\frac{1}{2}}} k. \lceil c\langle 1 \rangle = k \rceil) \\ \vdash & C_{\text{Ber}_p} m. C_{\text{Ber}_{\frac{1}{2}}} k. (\lceil m\langle 1 \rangle = m \rceil * \lceil c\langle 1 \rangle = k \rceil) \quad (\text{C-FRAME}) \\ \vdash & C_{\text{Ber}_p} m. C_{\text{Ber}_{\frac{1}{2}}} k. \lceil m\langle 1 \rangle = m \wedge c\langle 1 \rangle = k \rceil \quad (\text{SURE-MERGE}) \\ \vdash & C_{\text{Ber}_p \otimes \text{Ber}_{\frac{1}{2}}} (m, k). \lceil (m\langle 1 \rangle, c\langle 1 \rangle) = (m, k) \rceil \quad (\text{C-ASSOC}) \\ \vdash & (m\langle 1 \rangle, c\langle 1 \rangle) \sim (\text{Ber}_p \otimes \text{Ber}_{\frac{1}{2}}) \quad (\text{C-UNIT-R}) \end{aligned}$$

<pre> def BelowMax(x, S): repeat N: q ≈ μ_S r' := r r := r' q ≥ x </pre>	<pre> def AboveMin(x, S): repeat N: p ≈ μ_S l' := l l := l' p ≤ x </pre>	<pre> def BETW_SEQ(x, S): BelowMax(x, S); AboveMin(x, S); d := r && l </pre>
<pre> def BETW(x, S): repeat 2N: s ≈ μ_S l' := l l := l' s ≤ x r' := r r := r' s ≥ x d := r && l </pre>	<pre> def BETW_MIX(x, S): repeat N: p ≈ μ_S l' := l l := l' p ≤ x q ≈ μ_S r' := r r := r' q ≥ x d := r && l </pre>	<pre> def BETW_N(x, S): repeat N: s ≈ μ_S l' := l l := l' s ≤ x r' := r r := r' s ≥ x d := r && l </pre>

Fig. 13. Stochastic dominance examples: composing Monte Carlo algorithms in different ways. All variables are initially 0.

$$\vdash m\langle 1 \rangle \sim \text{Ber}_p * c\langle 1 \rangle \sim \text{Ber}_{1/2} \quad (\text{PROD-SPLIT})$$

The application of **C-TRANSF** to the case with $m = 1$ is as follows:

$$\frac{\forall b \in \{0, 1\}. \text{Ber}_{1/2}(b) = \text{Ber}_{1/2}(\neg b)}{C_{\text{Ber}_{1/2}} k. \lceil c\langle 1 \rangle = \neg k \rceil \vdash C_{\text{Ber}_{1/2}} k. \lceil c\langle 1 \rangle = k \rceil}$$

G.4 Markov Blanket and Variable Elimination

In probabilistic reasoning, introducing conditioning is easy, but deducing unconditional facts from conditional ones is not immediate. The same applies to the joint conditioning modality: by design, one cannot eliminate it for free. Crucial to BLUEBELL's expressiveness is the inclusion of rules that can soundly derive unconditional information from conditional assertions.

In this example we show how BLUEBELL is able to derive a common tool used in Bayesian reasoning to simplify conditioning as much as possible through the concept of a *Markov Blanket*.

For concreteness, consider the program:

$$x_1 \approx d_1; \quad x_2 \approx d_2(x_1); \quad x_3 \approx d_3(x_2)$$

The program describes a Markov chain of three variables. One way of interpreting this pattern is that the joint output distribution is described by the program as a product of conditional distributions: the distribution over x_2 is described conditionally on x_1 , and the one of x_3 conditionally on x_2 . This kind of dependencies are ubiquitous in, for instance, hidden Markov models and Bayesian network representations of distributions.

A crucial tool for the analysis of such models is the concept of a *Markov Blanket* of a variable x : the set of variables that are direct dependencies of x . Clearly x_3 depends on x_2 and, indirectly, on x_1 . However, Markov chains enjoy the memorylessness property: when fixing a variable in the chain, the variables that follow it are independent from the variables that preceded it. For our example this means that if we condition on x_2 , x_1 and x_3 are independent (i.e. we can ignore the indirect dependencies).

In BLUEBELL we can characterize the output distribution with the assertion

$$C_{d_1} v_1. \left(\lceil x1 = v_1 \rceil * C_{d_2(v_1)} v_2. (\lceil x2 = v_2 \rceil * x3 \sim d_3(v_2)) \right)$$

Note how this postcondition represents the output distribution as implicitly as the program does. We want to transform the assertion into:

$$C_{\mu_2} v_2. (\lceil x2 = v_2 \rceil * x1 \sim \mu_1(v_2) * x3 \sim d_3(v_2))$$

for appropriate μ_2 and μ_1 . This isolates the conditioning to the direct dependency of $x1$ and keeps full information about $x3$, available for further manipulation down the line.

In probability theory, the proof of memorylessness is an application of Bayes' law: we are computing the distribution of $x1$ conditioned on $x2$, from the distribution of $x2$ conditioned on $x1$.

In BLUEBELL we can produce the transformation using the joint conditioning rules:

$$\begin{aligned} & C_{d_1} v_1. \left(\lceil x1 = v_1 \rceil * C_{d_2(v_1)} v_2. (\lceil x1 = v_2 \rceil * x3 \sim d_3(v_2)) \right) \\ \vdash & C_{d_1} v_1. \left(C_{d_2(v_1)} v_2. (\lceil x1 = v_1 \rceil * \lceil x1 = v_2 \rceil * x3 \sim d_3(v_2)) \right) & \text{(C-FRAME)} \\ \vdash & C_{\mu_0}(v_1, v_2). (\lceil x1 = v_1 \rceil * \lceil x2 = v_2 \rceil * x3 \sim d_3(v_2)) & \text{(C-ASSOC)} \\ \vdash & C_{\mu_2} v_2. \left(C_{\mu_1(v_2)} v_1. (\lceil x1 = v_1 \rceil * \lceil x2 = v_2 \rceil * x3 \sim d_3(v_2)) \right) & \text{(C-UNASSOC)} \\ \vdash & C_{\mu_2} v_2. \left(\lceil x2 = v_2 \rceil * C_{\mu_1(v_2)} v_1. (\lceil x1 = v_1 \rceil * x3 \sim d_3(v_2)) \right) & \text{(SURE-STR-CONVEX)} \\ \vdash & C_{\mu_2} v_2. (\lceil x2 = v_2 \rceil * x1 \sim \mu_1(v_2) * x3 \sim d_3(v_2)) & \text{(C-EXTRACT)} \end{aligned}$$

where $(d_1 \prec d_2) = \mu_0 = \text{bind}(\mu_2, \mu_1)$. The existence of such μ_2 and μ_1 is a simple application of Bayes' law: $\mu_2(v_2) = \sum_{v_1 \in \mathbb{V}} \mu_0(v_1, v_2)$, and $\mu_1(v_2)(v_1) = \frac{\mu_0(v_1, v_2)}{\mu_2(v_2)}$.

G.5 Multi-party Secure Computation

The idea of *multi-party secure computation* is to allow N parties to compute a function $f(x_1, \dots, x_N)$ of some private data x_i owned by each party i , without revealing any more information about x_i than the output of f would reveal if computed centrally by a trusted party. When f is addition, a secure computation of f is useful, for example, to compute the total number of votes without revealing who voted positively: some information would leak (e.g., if the total is non-zero then *somebody* voted positively) but only what is revealed by knowing the total and nothing more.

To achieve this objective, multi-party secure addition (MPSAdd) works by having the parties break their secret into N *secret shares* which individually look random, but the sum of which amounts to the original secret. These secret shares are then distributed to the other parties so that each party knows an incomplete set of shares of the other parties. Yet, each party can reliably compute the result of the function by computing a function of the received shares.

Figure 14 shows the MPSAdd algorithm, for the case $N = 3$, as modelled in [Barthe et al. 2019]. The algorithm works as follows. Each party i knows its secret x_i . All the parties agree on an appropriate prime number p to use, and want to compute the sum of all the secrets (modulo p). The algorithm goes through three phases:

- *Phase 1:* Each party i computes its three secret shares in row $r[i][\cdot]$ by generating two independent random numbers $r[i][1]$, $r[i][2]$ in \mathbb{Z}_p drawing from the *uniform* distribution \mathbb{U}_p . The third share $r[i][3]$ is chosen so that the sum of the shares amounts to the secret x_i .
Then columns are communicated so that each party i receives column $r[\cdot][j]$ for every $j \neq i$. Each party now knows 2 out of 3 of the shares of each other party.

```

1  def MPSAdd:
2    // Phase 1
3    for i in [1,2,3]:
4      r[i][1] := Up // Uniform sample from  $\mathbb{Z}_p$ 
5      r[i][2] := Up
6      r[i][3] := xi - r[i][1] - r[i][2] mod p
7    // Phase 2
8    for i in [1,2,3]:
9      s[i] := r[1][i] + r[2][i] + r[3][i] mod p
10   // Phase 3
11   sum := s[1] + s[2] + s[3] mod p

```

Fig. 14. Multi-party secure addition.

- *Phase 2*: Each party i computes, for every $j \neq i$, $s[j]$ as the sum of the column $r[-][j]$, and sends it to the other parties.
- *Phase 3*: Each party now knows $s[-]$, the sum of which is the sum of the secrets.

Barthe et al. [2019] provide a partial proof of the example: (1) They only verify uniformity and independence from the input of the secret shares r (with a proof that involves ad-hoc axioms); (2) As PSL can only represent (unconditional) independence, the proof cannot state anything useful about the other values s that are circulated to compute the sum; in principle they can also leak too much information, but will not be independent of the secret since they are supposed to disclose their sum.

In this section we set out to prove the stronger property that, by the end of the computation, nothing is revealed by the values party i received from the other parties other than the sum. We focus on party 1 as the specifications and proofs for the other parties are entirely analogous.

We observe that the above goal can be formalized using two very different judgments, one unary and one relational.

The **unary specification** says that, conditionally on the secret of party i , and the sum of the other secrets, all the values received by i (we call this the *view* of i) are independent from the secrets of the other parties; moreover the learned components of r are uniformly distributed.

Formally, the view of party 1 is the vector:

$$\text{view}_1 = (r[1][-], r[2][2], r[2][3], r[3][2], r[3][3], s[-], \text{sum})$$

The unary specification would then assume an arbitrary distribution μ_0 of the secrets (making no assumption about their independence), and asserting in the postcondition that view_1 and (x_1, x_2) are independent conditionally on x_1 and $x_2 + x_3$ (i.e. conditionally on the secret of party 1 and the total sum of the secrets).

$$((x_1, x_2, x_3) \sim \mu_0) @ p \vdash \mathbf{wp} \text{MPSAdd} \left\{ C_{\tilde{\mu}_0}(v_1, v_{23}). \left(\left[x_1 = v_1 \wedge (x_2 + x_3) = v_{23} \right] * \left(\text{view}_1 \sim U(v_1, v_{23}) * \exists \mu_{23}. (x_2, x_3) \sim \mu_{23} \right) \right) \right\} \quad (79)$$

For readability we omit the indices on the term and variables as they are $\langle 1 \rangle$ everywhere; we also implicitly interpret equalities and sums to be modulo p . Here $U(v_1, v_{23})$ distributes the components of r in view_1 uniformly (except for $r[1][3]$), and $\tilde{\mu}_0 = ((x_1, x_2, x_3) \leftarrow \mu_0; \text{return}(x_1, x_2 + x_3))$;

moreover p contains all the necessary permissions for the assignments in MPSAdd. Note how the conditioning on the sum represents the expected leakage of the multi-party computation.

The **relational specification** says that when running the program from two initial states differing only in the secrets of the other parties, but not in their sum, the views of party i would be distributed in the same way. As a binary judgement, the specification can be formalized as:

$$\left[\begin{array}{l} x_1\langle 1 \rangle = x_1\langle 2 \rangle \\ (x_2 + x_3)\langle 1 \rangle = (x_2 + x_3)\langle 2 \rangle \end{array} \right] @p \vdash \text{wp} \left[\begin{array}{l} \textcolor{red}{1}: \text{MPSAdd} \\ \textcolor{red}{2}: \text{MPSAdd} \end{array} \right] \left\{ \begin{array}{l} x_1\langle 1 \rangle = x_1\langle 2 \rangle \\ (x_2 + x_3)\langle \textcolor{red}{1} \rangle = (x_2 + x_3)\langle \textcolor{red}{2} \rangle \\ \text{view}_1\langle 1 \rangle = \text{view}_1\langle 2 \rangle \end{array} \right\} \quad (80)$$

As a first result, we show that BLUEBELL can produce a proof for both specifications, one in unary style, the other in a relational-lifting style. This demonstrates how BLUEBELL supports both styles uniformly, making it possible to pick and choose the one that fits the prover's intuition best, or that turns out to be easier to carry out.

Having two very different specification for the same property, however, begs the question of whether the two specifications are really equivalent; moreover, we would like to make the choice of how to *prove* the property independent of how the property is represented. This is important, for example, because one proof strategy (e.g. the relational) might be more convenient for some program, but we might then want to reuse the proven specification in a larger proof that might be conducted in a different style (e.g. unary). Our second key result is that we show how we can derive one spec from the other *within* BLUEBELL, thus showing that picking a style for proving the program does not inhibit the reuse of the result in a different style. This also illustrates the fitness of BLUEBELL as a tool for abstract meta-level reasoning: in pRHL or PSL/Lilac, the adequacy of a specification (or conversion between styles) needs to be proven outside of the logic. In BLUEBELL this can happen within the same logic that supports the proofs of the programs.

The rest of the section is devoted to substantiating these claims, providing BLUEBELL proofs for:

- (1) the unary specification;
- (2) the relational specification (independently of the unary proof);
- (3) the equivalence of the two specifications.

Although the third item would spare us from proving one of the first two, we provide direct proofs in the two styles to provide a point of comparison between them.

G.5.1 Proof of the unary specification. As a first step, we can apply the rules for loops and assignments to obtain the postcondition Q :

$$\begin{aligned} Q &= X * R_{12} * R_3 * S * \text{Sum} & R_{12} &= *_{i \in \{1,2,3\}} (r[i][1] \sim \textcolor{violet}{U}_p * r[i][2] \sim \textcolor{violet}{U}_p) \\ X &= (x_1, x_2, x_3) \sim \mu_0 & R_3 &= *_{i \in \{1,2,3\}} \left[r[i][3] = x_i - r[i][1] - r[i][2] \right] \\ \text{Sum} &= \left[\text{sum} = s[1] + s[2] + s[3] \right] & S &= \left[\bigwedge_{i \in \{1,2,3\}} s[i] = r[1][i] + r[2][i] + r[3][i] \right] \end{aligned}$$

Now the goal is to show that Q entails the postcondition of (79). As a first step we transform X into $(x_1, x_2 + x_3) \sim \mu$ by **DIST-FUN**. Then we condition on $(x_1, x_2 + x_3, x_2x_3)$ and the variables in

R_{12} , obtaining:

$$C_{\mu'}(v_1, v_{23}, v_2, v_3). \left[\begin{array}{l} \left[x_1 = v_1 \wedge (x_2 + x_3) = v_{23} \wedge (x_2, x_3) = (v_2, v_3) \right] * \\ C_{U_p} u_{11} \cdot C_{U_p} u_{12} \cdot C_{U_p} u_{21} \cdot C_{U_p} u_{22} \cdot C_{U_p} u_{31} \cdot C_{U_p} u_{32} \cdot \\ \left[\begin{array}{l} r[1][1] = u_{11} \wedge r[1][2] = u_{12} \wedge r[1][3] = v_1 - u_{11} - u_{12} \\ r[2][2] = u_{22} \wedge r[2][3] = v_2 - u_{21} - u_{22} \\ r[3][2] = u_{32} \wedge r[3][3] = (v_{23} - v_2) - u_{31} - u_{32} \\ s[1] = u_{11} + u_{21} + u_{31} \\ s[2] = u_{12} + u_{22} + u_{32} \\ s[3] = v_1 - u_{11} - u_{12} + v_2 - u_{21} - u_{22} + (v_{23} - v_2) - u_{31} - u_{32} \\ \text{sum} = s[1] + s[2] + s[3] \end{array} \right] \end{array} \right]$$

Here $\mu' = (x_1, x_2, x_3) \leftarrow \mu_0$; **return** $(x_1, x_2 + x_3, x_2)$. We already weakened the assertion by forgetting the information about $r[2][1]$ and $r[3][1]$, which are not part of view_1 .

Now we perform a change of variables thanks to Section 4.3, to express our equalities in terms of $u'_{21} = u_{21} - v_2$ instead of u_{21} and $u'_{31} = u_{31} - (v_{23} - v_2)$ instead of u_{31} . To justify the change we simply observe that, for all $n \in \mathbb{Z}_p$, the function $f_n(u) = u - n \bmod p$ is a bijection and $U_p \circ f_n^{-1} = U_p$. This gives us, with some simple arithmetic simplifications:

$$C_{\mu'}(v_1, v_{23}, v_2, v_3). \left[\begin{array}{l} \left[x_1 = v_1 \wedge (x_2 + x_3) = v_{23} \wedge (x_2, x_3) = (v_2, v_3) \right] * \\ C_{U_p} u_{11} \cdot C_{U_p} u_{12} \cdot C_{U_p} u'_{21} \cdot C_{U_p} u_{22} \cdot C_{U_p} u'_{31} \cdot C_{U_p} u_{32} \cdot \\ \left[\begin{array}{l} r[1][1] = u_{11} \wedge r[1][2] = u_{12} \wedge r[1][3] = v_1 - u_{11} - u_{12} \\ r[2][2] = u_{22} \wedge r[2][3] = -u'_{21} - u_{22} \\ r[3][2] = u_{32} \wedge r[3][3] = -u'_{31} - u_{32} \\ s[1] = u_{11} + u'_{21} + u'_{31} + v_{23} \\ s[2] = u_{12} + u_{22} + u_{32} \\ s[3] = v_1 - u_{11} - u_{12} - u'_{21} - u_{22} - u'_{31} - u_{32} \\ \text{sum} = s[1] + s[2] + s[3] \end{array} \right] \end{array} \right]$$

In particular we removed all dependencies on v_2 from the inner formula. We can now apply **C-ASSOC** to collapse all the inner conditioning into a single one:

$$C_{\mu'}(v_1, v_{23}, v_2, v_3). \left(\left[x_1 = v_1 \wedge (x_2 + x_3) = v_{23} \wedge (x_2, x_3) = (v_2, v_3) \right] * \right. \\ \left. C_{U(v_1, v_{23})} \mathbf{u}. [\text{view}_1 = \mathbf{u}] \right)$$

where $U(v_1, v_{23}) = (v \leftarrow U_p \otimes \dots \otimes U_p; \text{return } g(v))$ takes the six independent samples from U_p and returns the values for each of the components of view_1 (which justifies the dependency on v_1 and v_{23}). Finally, we split $\mu' = \text{bind}(\mu, \kappa)$ obtaining:

$$C_{\mu'}(v_1, v_{23}, v_2, v_3). \left(\left[x_1 = v_1 \wedge (x_2 + x_3) = v_{23} \wedge (x_2, x_3) = (v_2, v_3) \right] * \right. \\ \left. C_{U(v_1, v_{23})} \mathbf{u}. [\text{view}_1 = \mathbf{u}] \right) \\ \vdash C_{\mu'}(v_1, v_{23}, v_2, v_3). \left(\left[x_1 = v_1 \wedge (x_2 + x_3) = v_{23} \right] * \right. \\ \left. \left[\begin{array}{l} (x_2, x_3) = (v_2, v_3) \\ \text{view}_1 \sim U(v_1, v_{23}) \end{array} \right] * \right) \quad (\text{SURE-MERGE, C-UNIT-R})$$

$$\begin{aligned}
& \vdash C_{\tilde{\mu}_0}(v_1, v_{23}) \cdot \left(\begin{array}{l} \lceil x_1 = v_1 \wedge (x_2 + x_3) = v_{23} \rceil * \\ C_{\kappa(v_1, v_{23})}(v_2, v_3) \cdot \\ \lceil (x_2, x_3) = (v_2, v_3) \rceil * \text{view}_1 \sim U(v_1, v_{23}) \end{array} \right) \quad (\text{C-UNASSOC, SURE-STR-CONVEX}) \\
& \vdash C_{\tilde{\mu}_0}(v_1, v_{23}) \cdot \left(\begin{array}{l} \lceil x_1 = v_1 \wedge (x_2 + x_3) = v_{23} \rceil * \\ (x_2, x_3) \sim \kappa(v_1, v_{23}) * \text{view}_1 \sim U(v_1, v_{23}) \end{array} \right) \quad (\text{C-EXTRACT}) \\
& \vdash C_{\tilde{\mu}_0}(v_1, v_{23}) \cdot \left(\begin{array}{l} \lceil x_1 = v_1 \wedge (x_2 + x_3) = v_{23} \rceil * \\ \text{view}_1 \sim U(v_1, v_{23}) * \exists \mu_{23}. (x_2, x_3) \sim \mu_{23} \end{array} \right)
\end{aligned}$$

This gets us the desired postcondition, and concludes the proof.

G.5.2 Proof of the relational specification. We now want to prove the goal (80) using the relational lifting technique, i.e. inducing a suitable coupling between the variables of the two components. Starting from precondition $\lfloor x_1 \langle 1 \rangle = x_1 \langle 2 \rangle \wedge (x_2 + x_3) \langle 1 \rangle = (x_2 + x_3) \langle 2 \rangle \rfloor$, we can proceed just like in the unary proof, blindly applying the loop and assignment rules obtaining (reusing the assertions of the previous section):

$$\left\lfloor \begin{array}{l} x_1 \langle 1 \rangle = x_1 \langle 2 \rangle \\ (x_2 + x_3) \langle 1 \rangle = (x_2 + x_3) \langle 2 \rangle \end{array} \right\rfloor * \left(\begin{array}{l} R_{12} \langle 1 \rangle * R_3 \langle 1 \rangle * S \langle 1 \rangle * \text{Sum} \langle 1 \rangle * \\ R_{12} \langle 2 \rangle * R_3 \langle 2 \rangle * S \langle 2 \rangle * \text{Sum} \langle 2 \rangle \end{array} \right)$$

Now the main task is to couple the sources of randomness in R_{12} so that the views of 1 coincide in the two components. The idea is that if we can induce a coupling where $r[2][1] \langle 1 \rangle = r[2][1] \langle 2 \rangle + (x_2 \langle 1 \rangle - x_2 \langle 2 \rangle)$ and $r[3][1] \langle 1 \rangle = r[3][1] \langle 2 \rangle + (x_3 \langle 1 \rangle - x_3 \langle 2 \rangle)$ then we would obtain $\text{view}_1 \langle 1 \rangle = \text{view}_1 \langle 2 \rangle$. To implement the idea we combine two observations: (1) we can induce the desired coupling if we are under conditioning of x_2 and x_3 on both indices; a conditioning that is already happening in the relational lifting of the precondition; (2) the coupling is valid because addition of some constant modulo p to a uniformly distributed variable, gives a uniformly distributed variable (an observation we also exploited in the unary proof).

Formally, we first unfold the definition of the relational lifting to reveal the joint conditioning on x_1, x_2 , and x_3 , and move the other resources inside the conditioning by **C-FRAME**:

$$\begin{aligned}
& \left\lfloor \begin{array}{l} x_1 \langle 1 \rangle = x_1 \langle 2 \rangle \\ (x_2 + x_3) \langle 1 \rangle = (x_2 + x_3) \langle 2 \rangle \end{array} \right\rfloor * \left(\begin{array}{l} R_{12} \langle 1 \rangle * R_3 \langle 1 \rangle * S \langle 1 \rangle * \text{Sum} \langle 1 \rangle * \\ R_{12} \langle 2 \rangle * R_3 \langle 2 \rangle * S \langle 2 \rangle * \text{Sum} \langle 2 \rangle \end{array} \right) \\
& \vdash \exists \hat{\mu}. C_{\hat{\mu}}(v_1, v_2, v_3) \cdot \left(\begin{array}{l} \lceil v_1 = w_1 \wedge v_2 + v_3 = w_2 + w_3 \rceil * \\ \lceil x_i \langle 1 \rangle = v_i \rceil_{i \in \{1,2,3\}} * \\ \lceil x_i \langle 2 \rangle = w_i \rceil_{i \in \{1,2,3\}} \end{array} \right) * \left(\begin{array}{l} R_{12} \langle 1 \rangle * R_3 \langle 1 \rangle * S \langle 1 \rangle * \text{Sum} \langle 1 \rangle * \\ R_{12} \langle 2 \rangle * R_3 \langle 2 \rangle * S \langle 2 \rangle * \text{Sum} \langle 2 \rangle \end{array} \right) \\
& \vdash \exists \hat{\mu}. C_{\hat{\mu}}(v_1, v_2, v_3) \cdot \left(\begin{array}{l} \lceil v_1 = w_1 \wedge v_2 + v_3 = w_2 + w_3 \rceil * \\ \lceil x_i \langle 1 \rangle = v_i \rceil_{i \in \{1,2,3\}} * R_{12} \langle 1 \rangle * R_3 \langle 1 \rangle * S \langle 1 \rangle * \text{Sum} \langle 1 \rangle * \\ \lceil x_i \langle 2 \rangle = w_i \rceil_{i \in \{1,2,3\}} * R_{12} \langle 2 \rangle * R_3 \langle 2 \rangle * S \langle 2 \rangle * \text{Sum} \langle 2 \rangle \end{array} \right) \quad (\text{C-FRAME})
\end{aligned}$$

Now by using **C-CONS**, we can induce a coupling inside the conditioning. By using **COUPLING** we obtain:

$$R_{12} \langle 1 \rangle * R_{12} \langle 2 \rangle \vdash \left(\begin{array}{l} *_{i \in \{1,2,3\}} (r[i][1] \langle 1 \rangle \sim U_p * r[i][2] \langle 1 \rangle \sim U_p) * \\ *_{i \in \{1,2,3\}} (r[i][1] \langle 2 \rangle \sim U_p * r[i][2] \langle 2 \rangle \sim U_p) \end{array} \right)$$

$$\vdash \left[\begin{array}{l} r[1][1]\langle 1 \rangle = r[1][1]\langle 2 \rangle \\ \bigwedge_{i \in \{1,2,3\}} r[i][2]\langle 1 \rangle = r[i][2]\langle 2 \rangle \\ r[2][1]\langle 1 \rangle = r[2][1]\langle 2 \rangle + (v_2 - w_2) \\ r[3][1]\langle 1 \rangle = r[3][1]\langle 2 \rangle + (v_3 - w_3) \end{array} \right] \quad (\text{COUPLING})$$

The application of **COUPLING** is supported by the coupling

$$\mu = (u_1, \dots, u_6) \leftarrow U_p^{(6)}; \text{ return } \left(\begin{array}{l} (u_1, u_2, u_3 + (v_2 - w_2), u_4, u_5 + (v_3 - w_3), u_6), \\ (u_1, u_2, u_3, u_4, u_5, u_6) \end{array} \right)$$

where $U_p^{(6)}$ is the independent product of 6 U_p . The joint distribution μ satisfies $\mu \circ \pi_1^{-1} = U_p^{(6)}$, $\mu \circ \pi_2^{-1} = U_p^{(6)}$; note that $U_p^{(6)}$ is the distribution of $(r[1][1], r[1][2], r[2][1], r[3][1], r[3][2])$ at both indices 1 and 2 (by R_{12}).

Now we can merge the relational lifting with the other almost sure facts we have under conditioning, simplify and apply **RL-CONVEX** to obtain the desired unconditional coupling:

$$\begin{aligned} & C_{\hat{\mu}} \left(\begin{array}{l} v_1, v_2, v_3 \\ w_1, w_2, w_3 \end{array} \right) \cdot \left[\begin{array}{l} \lceil v_1 = w_1 \wedge v_2 + v_3 = w_2 + w_3 \rceil * \\ \lceil x_i \langle 1 \rangle = v_i \rceil_{i \in \{1,2,3\}} * R_{12} \langle 1 \rangle * R_3 \langle 1 \rangle * S \langle 1 \rangle * \text{Sum} \langle 1 \rangle * \\ \lceil x_i \langle 2 \rangle = w_i \rceil_{i \in \{1,2,3\}} * R_{12} \langle 2 \rangle * R_3 \langle 2 \rangle * S \langle 2 \rangle * \text{Sum} \langle 2 \rangle \end{array} \right] \\ & \vdash C_{\hat{\mu}} \left(\begin{array}{l} v_1, v_2, v_3 \\ w_1, w_2, w_3 \end{array} \right) \cdot \left[\begin{array}{l} \lceil v_1 = w_1 \wedge v_2 + v_3 = w_2 + w_3 \rceil * \\ \lceil x_i \langle 1 \rangle = v_i \rceil_{i \in \{1,2,3\}} * R_3 \langle 1 \rangle * S \langle 1 \rangle * \text{Sum} \langle 1 \rangle * \\ \lceil x_i \langle 2 \rangle = w_i \rceil_{i \in \{1,2,3\}} * R_3 \langle 2 \rangle * S \langle 2 \rangle * \text{Sum} \langle 2 \rangle * \\ \left[\begin{array}{l} r[1][1]\langle 1 \rangle = r[1][1]\langle 2 \rangle \\ \bigwedge_{i \in \{1,2,3\}} r[i][2]\langle 1 \rangle = r[i][2]\langle 2 \rangle \\ r[2][1]\langle 1 \rangle = r[2][1]\langle 2 \rangle + (v_2 - w_2) \\ r[3][1]\langle 1 \rangle = r[3][1]\langle 2 \rangle + (v_3 - w_3) \end{array} \right] \end{array} \right] \\ & \vdash C_{\hat{\mu}} \left(\begin{array}{l} v_1, v_2, v_3 \\ w_1, w_2, w_3 \end{array} \right) \cdot \left[\begin{array}{l} v_1 = w_1 \wedge v_2 + v_3 = w_2 + w_3 \\ r[1][1]\langle 1 \rangle = r[1][1]\langle 2 \rangle \\ \bigwedge_{i \in \{1,2,3\}} r[i][2]\langle 1 \rangle = r[i][2]\langle 2 \rangle \\ r[2][1]\langle 1 \rangle = r[2][1]\langle 2 \rangle + (v_2 - w_2) \\ r[3][1]\langle 1 \rangle = r[3][1]\langle 2 \rangle + (v_3 - w_3) \\ \bigwedge_{i \in \{1,2,3\}} x_i \langle 1 \rangle = v_i \wedge x_i \langle 2 \rangle = w_i \\ \bigwedge_{i \in \{1,2,3\}} (r[i][3] = x_i - r[i][1] - r[i][2]) \langle 1 \rangle \\ \dots \end{array} \right] \quad (\text{RL-SURE-MERGE}) \\ & \vdash C_{\hat{\mu}} \left(\begin{array}{l} v_1, v_2, v_3 \\ w_1, w_2, w_3 \end{array} \right) \cdot \left[\begin{array}{l} x_1 \langle 1 \rangle = x_1 \langle 2 \rangle \\ (x_2 + x_3) \langle 1 \rangle = (x_2 + x_3) \langle 2 \rangle \\ \text{view}_1 \langle 1 \rangle = \text{view}_1 \langle 2 \rangle \end{array} \right] \quad (\text{RL-CONS}) \\ & \vdash \left[\begin{array}{l} x_1 \langle 1 \rangle = x_1 \langle 2 \rangle \\ (x_2 + x_3) \langle 1 \rangle = (x_2 + x_3) \langle 2 \rangle \\ \text{view}_1 \langle 1 \rangle = \text{view}_1 \langle 2 \rangle \end{array} \right] \quad (\text{RL-CONVEX}) \end{aligned}$$

G.5.3 Proof of equivalence of the two specifications. Now we prove that one can prove the relational specification from the unary one, and vice versa.

From unary to relational. We want to show that assuming the unary specification (79) holds, we can derive the relational specification (80). We first need to bridge the gap between having one component in the assumption and two in the consequence. This is actually easy: the proof of

(79) is completely parametric in the index chosen for the program, so the same proof can prove two specifications, one where the index of the term is **1** and one where it is **2**. As a side note, this “reindexing” argument can be made into a rule of the logic following the LHC approach [D’Oswaldo et al. 2022] but we do not do this in this paper so we can focus on the novel aspects of the logic. More formally, let $P(\mu_0)$ and $Q(\mu_0)$ be the precondition and the postcondition of the unary specification (79). Furthermore, let $\llbracket R_1 \rrbracket$ and $\llbracket R_2 \rrbracket$ be the precondition and postcondition of the relational specification (80).

First we can infer a binary spec from two unary instances of the unary spec, for arbitrary $\mu_1, \mu_2 : \mathbb{D}(\mathbb{Z}_p^3)$:

$$\frac{P(\mu_1)\langle 1 \rangle \vdash \mathbf{wp} \llbracket 1 : \text{MPSAdd} \rrbracket \{Q(\mu_1)\langle 1 \rangle\} \quad P(\mu_2)\langle 2 \rangle \vdash \mathbf{wp} \llbracket 2 : \text{MPSAdd} \rrbracket \{Q(\mu_2)\langle 2 \rangle\}}{\left(\frac{P(\mu_1)\langle 1 \rangle}{P(\mu_2)\langle 2 \rangle} \right) \vdash \mathbf{wp} \left[\begin{array}{l} 1 : \text{MPSAdd} \\ 2 : \text{MPSAdd} \end{array} \right] \left\{ \begin{array}{l} Q(\mu_1)\langle 1 \rangle \\ Q(\mu_2)\langle 2 \rangle \end{array} \right\}} \quad (81)$$

Recalling from (79) that $\tilde{\mu} = ((x_1, x_2, x_3) \leftarrow \mu; \mathbf{return}(x_1, x_2 + x_3))$, the proof works by showing:

$$\llbracket R_1 \rrbracket \vdash \exists \mu_1, \mu_2. (P(\mu_1)\langle 1 \rangle \wedge P(\mu_2)\langle 2 \rangle) * \ulcorner \tilde{\mu}_1 = \tilde{\mu}_2 \urcorner \quad (82)$$

$$Q(\mu_1)\langle 1 \rangle \wedge Q(\mu_2)\langle 2 \rangle * \ulcorner \tilde{\mu}_1 = \tilde{\mu}_2 \urcorner \vdash \llbracket R_2 \rrbracket \quad (83)$$

From (82) we obtain μ_1 and μ_2 with which to instantiate (81), and that the precondition of (81) holds. Then, by applying **WP-CONS** to the conclusion of (81) and (83) we obtain the desired relational specification (80).

Entailment (82) is obtained in the same way one proves **RL-EQ-DIST**:

$$\begin{aligned} \left[\begin{array}{l} x_1\langle 1 \rangle = x_1\langle 2 \rangle \\ (x_2 + x_3)\langle 1 \rangle = (x_2 + x_3)\langle 2 \rangle \end{array} \right] &\vdash \exists \hat{\mu}. \ulcorner \hat{\mu}(R_1) = 1 \urcorner * C_{\hat{\mu}}\left(\left(\begin{array}{l} v_1, v_2, v_3 \\ w_1, w_2, w_3 \end{array}\right)\right). \left(\left[\begin{array}{l} x_i\langle 1 \rangle = v_i \\ x_i\langle 2 \rangle = w_i \end{array} \right]_{i \in \{1,2,3\}} \right) \\ &\vdash \exists \hat{\mu}. \ulcorner \hat{\mu}(R_1) = 1 \urcorner * \left(C_{\hat{\mu}}\left(\left(\begin{array}{l} v_1, v_2, v_3 \\ w_1, w_2, w_3 \end{array}\right)\right). \left[\begin{array}{l} x_i\langle 1 \rangle = v_i \\ x_i\langle 2 \rangle = w_i \end{array} \right]_{i \in \{1,2,3\}} \right) \\ &\vdash \exists \hat{\mu}. \ulcorner \hat{\mu}(R_1) = 1 \urcorner * \left(C_{\hat{\mu} \circ \pi_1^{-1}}(v_1, v_2, v_3). \left[\begin{array}{l} x_i\langle 1 \rangle = v_i \\ x_i\langle 2 \rangle = w_i \end{array} \right]_{i \in \{1,2,3\}} \right) \\ &\vdash \exists \mu_1, \mu_2. \left(\begin{array}{l} (x_1, x_2, x_3)\langle 1 \rangle \sim \mu_1 \\ (x_1, x_2, x_3)\langle 2 \rangle \sim \mu_2 \end{array} \right) * \ulcorner \tilde{\mu}_1 = \tilde{\mu}_2 \urcorner \end{aligned}$$

The last step is justified by letting $\mu_1 = \hat{\mu} \circ \pi_1^{-1}$ and $\mu_2 = \hat{\mu} \circ \pi_2^{-1}$, noting that $\hat{\mu}(R_1) = 1$ implies $\tilde{\mu}_1 = \tilde{\mu}_2$.

Finally, we prove (83). Under the assumption $\tilde{\mu}_1 = \tilde{\mu}_2$ we know that there is some coupling $\hat{\mu}$ such that $\hat{\mu} \circ \pi_1^{-1} = \tilde{\mu}_1 = \tilde{\mu}_2 = \hat{\mu} \circ \pi_2^{-1}$, and $\hat{\mu}(R) = 1$ where $R = \{((v_1, v_{23}), (v_1, v_{23})) \mid v_1, v_{23} \in \mathbb{Z}_p\}$.

$$\begin{aligned} &\left(\begin{array}{l} C_{\tilde{\mu}_1}(v_1, v_{23}). \left(\left[\begin{array}{l} x_1\langle 1 \rangle = v_1 \wedge (x_2 + x_3)\langle 1 \rangle = v_{23} \end{array} \right] * \right. \\ \quad \left. \text{view}_1\langle 1 \rangle \sim U(v_1, v_{23}) * \exists \mu_{23}. (x_2, x_3)\langle 1 \rangle \sim \mu_{23} \right) \\ C_{\tilde{\mu}_2}(w_1, w_{23}). \left(\left[\begin{array}{l} x_1\langle 2 \rangle = w_1 \wedge (x_2 + x_3)\langle 2 \rangle = w_{23} \end{array} \right] * \right. \\ \quad \left. \text{view}_1\langle 2 \rangle \sim U(w_1, w_{23}) * \exists \mu_{23}. (x_2, x_3)\langle 2 \rangle \sim \mu_{23} \right) \end{array} \right) \\ &\vdash \left(\begin{array}{l} C_{\hat{\mu}}\left(\begin{array}{l} v_1, v_{23} \\ w_1, w_{23} \end{array}\right). \left(\left[\begin{array}{l} x_1\langle 1 \rangle = v_1 \wedge (x_2 + x_3)\langle 1 \rangle = v_{23} \end{array} \right] * \right. \\ \quad \left. \text{view}_1\langle 1 \rangle \sim U(v_1, v_{23}) * \exists \mu_{23}. (x_2, x_3)\langle 1 \rangle \sim \mu_{23} \right) \\ C_{\hat{\mu}}\left(\begin{array}{l} v_1, v_{23} \\ w_1, w_{23} \end{array}\right). \left(\left[\begin{array}{l} x_1\langle 2 \rangle = w_1 \wedge (x_2 + x_3)\langle 2 \rangle = w_{23} \end{array} \right] * \right. \\ \quad \left. \text{view}_1\langle 2 \rangle \sim U(w_1, w_{23}) * \exists \mu_{23}. (x_2, x_3)\langle 2 \rangle \sim \mu_{23} \right) \end{array} \right) \quad (\text{C-SURE-PROJ}) \end{aligned}$$

$$\begin{aligned}
& \vdash C_{\hat{\mu}} \left(\begin{array}{l} \lceil x_1 \langle 1 \rangle = v_1 \wedge (x_2 + x_3) \langle 1 \rangle = v_{23} \rceil \\ \lceil x_1 \langle 2 \rangle = w_1 \wedge (x_2 + x_3) \langle 2 \rangle = w_{23} \rceil \\ \text{view}_1 \langle 1 \rangle \sim U(v_1, v_{23}) * \exists \mu_{23}. (x_2, x_3) \langle 1 \rangle \sim \mu_{23} \\ \text{view}_1 \langle 2 \rangle \sim U(w_1, w_{23}) * \exists \mu_{23}. (x_2, x_3) \langle 2 \rangle \sim \mu_{23} \\ \lceil v_1 = w_1 \wedge v_{23} = w_{23} \rceil \end{array} \right) \quad (\text{C-AND}) \\
& \vdash \exists \kappa_1, \kappa_2. C_{\hat{\mu}} \left(\begin{array}{l} \lceil x_1 \langle 1 \rangle = v_1 \wedge (x_2 + x_3) \langle 1 \rangle = v_{23} \rceil \\ \lceil x_1 \langle 2 \rangle = w_1 \wedge (x_2 + x_3) \langle 2 \rangle = w_{23} \rceil \\ \text{view}_1 \langle 1 \rangle = \text{view}_1 \langle 2 \rangle \\ (x_2, x_3) \langle 1 \rangle \sim \kappa_1(v_1, v_{23}, w_1, w_{23}) \\ (x_2, x_3) \langle 2 \rangle \sim \kappa_2(v_1, v_{23}, w_1, w_{23}) \\ \lceil v_1 = w_1 \wedge v_{23} = w_{23} \rceil \end{array} \right) \quad (\text{C-SKOLEM, COUPLING}) \\
& \vdash \exists \kappa_1, \kappa_2. C_{\hat{\mu}} \left(\begin{array}{l} \lceil x_1 \langle 1 \rangle = v_1 \wedge (x_2 + x_3) \langle 1 \rangle = v_{23} \rceil \\ \lceil x_1 \langle 2 \rangle = w_1 \wedge (x_2 + x_3) \langle 2 \rangle = w_{23} \rceil \\ \text{view}_1 \langle 1 \rangle = \text{view}_1 \langle 2 \rangle \\ C_{\kappa_1(v_1, v_{23}, w_1, w_{23})}(v_2, v_3). \lceil (x_2, x_3) \langle 1 \rangle = (v_1, v_2) \rceil \\ C_{\kappa_2(v_1, v_{23}, w_1, w_{23})}(w_2, w_3). \lceil (x_2, x_3) \langle 2 \rangle = (w_1, w_2) \rceil \\ \lceil v_1 = w_1 \wedge v_{23} = w_{23} \rceil \end{array} \right) \quad (\text{C-UNIT-R}) \\
& \vdash \exists \hat{\mu}_1. C_{\hat{\mu}_1} \left(\begin{array}{l} \lceil x_1 \langle 1 \rangle = v_1 \wedge (x_2 + x_3) \langle 1 \rangle = v_{23} \rceil \\ \lceil x_1 \langle 2 \rangle = w_1 \wedge (x_2 + x_3) \langle 2 \rangle = w_{23} \rceil \\ \text{view}_1 \langle 1 \rangle = \text{view}_1 \langle 2 \rangle \\ \lceil (x_2, x_3) \langle 1 \rangle = (v_1, v_2) \rceil \\ \lceil (x_2, x_3) \langle 2 \rangle = (w_1, w_2) \rceil \\ \lceil v_1 = w_1 \wedge v_{23} = w_{23} \rceil \end{array} \right) \quad (\text{C-FRAME, C-ASSOC}) \\
& \vdash \exists \hat{\mu}_2. C_{\hat{\mu}_2} \left(\begin{array}{l} \lceil x_i \langle 1 \rangle = v_i \rceil_{i \in \{1,2,3\}} \wedge \lceil x_i \langle 2 \rangle = w_i \rceil_{i \in \{1,2,3\}} \\ \text{view}_1 \langle 1 \rangle = \text{view}_1 \langle 2 \rangle \\ \lceil v_1 = w_1 \wedge v_2 + v_3 = w_2 + w_3 \rceil \end{array} \right) \quad (\text{C-TRANSF}) \\
& \vdash \exists \hat{\mu}_2. C_{\hat{\mu}_2}(v, w). \left(\begin{array}{l} \lceil x_i \langle 1 \rangle = v_i \rceil_{i \in \{1,2,3\}} \wedge \lceil x_i \langle 2 \rangle = w_i \rceil_{i \in \{1,2,3\}} \\ \exists \hat{v}. C_{\hat{v}}(u_1, u_2). \lceil \text{view}_1 \langle 1 \rangle = u_1 \rceil \wedge \lceil \text{view}_1 \langle 2 \rangle = u_2 \rceil \wedge \lceil u_1 = u_2 \rceil \\ \lceil v_1 = w_1 \wedge v_2 + v_3 = w_2 + w_3 \rceil \end{array} \right) \quad (\text{By def.}) \\
& \vdash \exists \hat{\mu}_3. C_{\hat{\mu}_3}(v, w, u_1, u_2). \left(\begin{array}{l} \lceil x_i \langle 1 \rangle = v_i \rceil_{i \in \{1,2,3\}} \wedge \lceil x_i \langle 2 \rangle = w_i \rceil_{i \in \{1,2,3\}} \\ \lceil \text{view}_1 \langle 1 \rangle = u_1 \rceil \wedge \lceil \text{view}_1 \langle 2 \rangle = u_2 \rceil \wedge \\ \lceil v_1 = w_1 \wedge v_2 + v_3 = w_2 + w_3 \wedge u_1 = u_2 \rceil \end{array} \right) \quad (\text{C-SKOLEM, C-ASSOC}) \\
& \vdash \left(\begin{array}{l} x_1 \langle 1 \rangle = x_1 \langle 2 \rangle \\ (x_2 + x_3) \langle 1 \rangle = (x_2 + x_3) \langle 2 \rangle \\ \text{view}_1 \langle 1 \rangle = \text{view}_1 \langle 2 \rangle \end{array} \right) \quad (\text{By def.})
\end{aligned}$$

From relational to unary. The proof in the reverse direction follows the same strategy as the unary to relational direction, except the entailments between preconditions and postconditions are reversed; their proof steps in the previous section are however reversible, so we do not repeat

the proof here. There is one niggle: what we can derive from the relational specification is the judgment $P(\mu_0)\langle 1 \rangle \wedge P(\mu_0)\langle 2 \rangle \vdash \mathbf{wp} [1:\text{MPSAdd}, 2:\text{MPSAdd}] \{Q(\mu_0)\langle 1 \rangle \wedge Q(\mu_0)\langle 2 \rangle\}$ which in fact implies the unary specification (by ignoring component 2), but we have not provided BLUEBELL with rules to eliminate components. The issue has been solved in LHC [D'Ossualdo et al. 2022] with the *projection modality* $\Pi_i. P = \lambda a. \exists b. P(a[i:b])$, which supports the principle

$$\frac{\text{WP-PROJ} \quad P \vdash \mathbf{wp} t \{Q\}}{\Pi_i. P \vdash \mathbf{wp} (t \setminus i) \{\Pi_i. Q\}}$$

The rule is sound in BLUEBELL model and could be used to fill this (small) gap since $P(\mu_0)\langle 1 \rangle \vdash \Pi_2. (P(\mu_0)\langle 1 \rangle \wedge P(\mu_0)\langle 2 \rangle)$ and $\Pi_2. (Q(\mu_0)\langle 1 \rangle \wedge Q(\mu_0)\langle 2 \rangle) \vdash Q(\mu_0)\langle 1 \rangle$. We did not emphasize projection in this paper to avoid distracting from the main novel contributions.

G.6 Von Neumann Extractor

A randomness extractor is a mechanism that transforms a stream of “low-quality” randomness sources into a stream of “high-quality” randomness sources. The Von Neumann extractor is perhaps the earliest instance of such mechanism, and it converts a stream of independent coins with the same bias p into a stream of independent *fair* coins.

In our language we can model the extractor, up to $N \in \mathbb{N}$ iterations, as shown in Fig. 5. The program repeatedly flips two biased coins, and outputs the outcome of the first coin if the outcomes where different, otherwise it retries. What we can prove is that the bits produced in out are independent fair coin flips: if we produced ℓ bits we should be able to prove the postcondition

$$\text{Out}_\ell \triangleq \text{out}[0]\langle 1 \rangle \sim \text{Ber}_{1/2} * \dots * \text{out}[\ell-1]\langle 1 \rangle \sim \text{Ber}_{1/2}.$$

To know how many bits were produced, however, we need to condition on len obtaining the specification:

$$\vdash \mathbf{wp} [1:\text{vn}(N)] \{\exists \mu. C_\mu \ell. (\lceil \text{len}\langle 1 \rangle = \ell \leq N \rceil * \text{Out}_\ell)\}$$

(Recall $P \Vdash Q \triangleq P \wedge \text{own}_{\mathbb{X}} \vdash Q \wedge \text{own}_{\mathbb{X}}$)

The BLUEBELL proof of this specification is shown in the outline in Fig. 15. The postcondition straightforwardly generalizes to a loop invariant

$$P(i) = \exists \mu. C_\mu \ell. (\lceil \text{len}\langle 1 \rangle = \ell \leq i \rceil * \text{Out}_\ell)$$

At step (84) we show, by using C-UNIT-I and the definition of $\lceil \cdot \rceil$, that we can obtain the loop invariant with $i = 0$: $P(0) = \exists \mu. C_\mu \ell. (\lceil \text{len}\langle 1 \rangle = \ell \leq 0 \rceil * \text{Out}_0) = \exists \mu. C_\mu \ell. (\lceil \text{len}\langle 1 \rangle = \ell \leq 0 \rceil)$.

For the proof of the body of the loop we can assume $P(i)$ and we need to prove the postcondition $P(i+1)$. After sampling the two coins, we reach the point where we apply the fundamental insight behind the extractor, at step (85). The key idea is that with some probability q the two coins will be different, in which case the outcomes of the two coins can be either (0, 1) or (1, 0), which both have the same probability $p(1-p)$. Therefore, if the coins are different, $\text{coin}_1 = 0$ and $\text{coin}_1 = 1$ have the same probability, i.e. coin_1 looks like a fair coin.

BLUEBELL is capable of representing this reasoning as follows.

We start with two independent biased coins, which we can combine into a random variable $(\text{coin}_1 \neq \text{coin}_2, \text{coin}_1)$ recording whether the two outcomes where different and the outcome of the first coin; it is easy to derive (using PROD-UNSPLIT and DIST-FUN):

$$\text{coin}_1\langle 1 \rangle \sim \text{Ber}_p * \text{coin}_2\langle 1 \rangle \sim \text{Ber}_p \vdash (\text{coin}_1 \neq \text{coin}_2, \text{coin}_1)\langle 1 \rangle \sim \mu_0$$

$$\begin{array}{l}
\{\text{own}_X\} \\
\text{len} := 0 \\
\{\lceil \text{len}\langle 1 \rangle = 0 \rceil\} \\
\{\exists \mu. C_\mu \ell. (\lceil \text{len}\langle 1 \rangle = \ell \leq 0 \rceil)\} \quad (84) \\
\text{repeat } N: \\
\quad \{P(i)\} \\
\quad \text{coin}_1 \approx \text{Ber}(p) \\
\quad \text{coin}_2 \approx \text{Ber}(p) \\
\quad \{P(i) * \text{coin}_1\langle 1 \rangle \sim \text{Ber}_p * \text{coin}_2\langle 1 \rangle \sim \text{Ber}_p\} \\
\quad \left\{ C_\mu \ell. C_\beta b. \left(\begin{array}{l} \lceil \text{len}\langle 1 \rangle = \ell \leq i \rceil * \lceil (\text{coin}_1 \neq \text{coin}_2)\langle 1 \rangle = b \rceil \\ * \text{Out}_\ell * (\lceil b = 1 \rceil \Rightarrow \text{coin}_1\langle 1 \rangle \sim \text{Ber}_{\frac{1}{2}}) \end{array} \right) \right\} \quad (85) \\
\quad \left\{ \begin{array}{l} \lceil \text{len}\langle 1 \rangle = \ell \leq i \rceil * \lceil (\text{coin}_1 \neq \text{coin}_2)\langle 1 \rangle = b \rceil \\ * \text{Out}_\ell * (\lceil b = 1 \rceil \Rightarrow \text{coin}_1\langle 1 \rangle \sim \text{Ber}_{\frac{1}{2}}) \end{array} \right\} \\
\quad \text{if } \text{coin}_1 \neq \text{coin}_2 \text{ then:} \\
\quad \quad \left\{ \begin{array}{l} \lceil \text{len}\langle 1 \rangle = \ell \leq i \rceil * \lceil (\text{coin}_1 \neq \text{coin}_2)\langle 1 \rangle \rceil \\ * \text{Out}_\ell * \text{coin}_1\langle 1 \rangle \sim \text{Ber}_{\frac{1}{2}} \end{array} \right\} \\
\quad \quad \text{out}[\text{len}] := \text{coin}_1 \\
\quad \quad \text{len} := \text{len} + 1 \\
\quad \quad \left\{ \begin{array}{l} \lceil \text{len}\langle 1 \rangle = \ell + 1 \leq i + 1 \rceil * \lceil (\text{coin}_1 \neq \text{coin}_2)\langle 1 \rangle \rceil \\ * \text{Out}_\ell * \text{coin}_1\langle 1 \rangle \sim \text{Ber}_{\frac{1}{2}} * \lceil \text{out}[\text{len}] = \text{coin}_1\langle 1 \rangle \rceil \end{array} \right\} \\
\quad \quad \left\{ \begin{array}{l} \lceil (\text{coin}_1 \neq \text{coin}_2)\langle 1 \rangle = b \rceil * \text{Out}_\ell \\ * \left\{ \begin{array}{l} \lceil \text{len}\langle 1 \rangle = \ell + 1 \leq i + 1 \rceil * \text{out}[\text{len}]\langle 1 \rangle \sim \text{Ber}_{\frac{1}{2}} \quad \text{if } b = 1 \\ \lceil \text{len}\langle 1 \rangle = \ell \leq i + 1 \rceil \quad \text{if } b = 0 \end{array} \right\} \end{array} \right\} \\
\quad \quad \left\{ C_\mu \ell. C_\beta b. \lceil (\text{coin}_1 \neq \text{coin}_2)\langle 1 \rangle = b \rceil * \text{Out}_\ell * \dots \right\} \\
\quad \quad \left\{ C_{\mu'} \ell'. (\lceil \text{len}\langle 1 \rangle = \ell' \leq i + 1 \rceil * \text{Out}_{\ell'}) \right\} \quad (86) \\
\quad \{\exists \mu. C_\mu \ell. (\lceil \text{len}\langle 1 \rangle = \ell \leq N \rceil * \text{Out}_\ell)\}
\end{array}$$

Fig. 15. Proof outline of the Von Neumann extractor example.

where $\mu_0 \triangleq (c_1 \leftarrow \text{Ber}_p; c_2 \leftarrow \text{Ber}_p; \text{return}(c_1 \neq c_2, c_1))$. Now the crucial observation of the extractor can be phrased as a reformulation of μ_0 :

$$\mu_0 = \beta \prec \kappa \quad \beta \triangleq \text{Ber}_q \quad \kappa(1) \triangleq \text{Ber}_{\frac{1}{2}} \quad \kappa(0) \triangleq \text{Ber}_{q'}$$

Here one first determines (with some probability q which is a function of p) whether the two coins will be different or equal, and then generates c_1 accordingly: in the “different” branch ($b = 1$) the first coin is distributed as $\text{Ber}_{\frac{1}{2}}$ while in the “equal” branch ($b = 0$) the first coin is distributed with some bias q' (also a function of p).

So using $\mu_0 = \beta \prec \kappa$ we derive:

$$\begin{array}{l}
(\text{coin}_1 \neq \text{coin}_2, \text{coin}_1)\langle 1 \rangle \sim (\beta \prec \kappa) \\
\vdash C_{\beta \prec \kappa}(b, c_1). \lceil (\text{coin}_1 \neq \text{coin}_2)\langle 1 \rangle = b \wedge \text{coin}_1\langle 1 \rangle = c_1 \rceil \quad (\text{C-UNIT-R}) \\
\vdash C_\beta b. C_{\kappa(b)} c_1. \lceil (\text{coin}_1 \neq \text{coin}_2)\langle 1 \rangle = b \rceil * \lceil \text{coin}_1\langle 1 \rangle = c_1 \rceil \quad (\text{C-FUSE}) \\
\vdash C_\beta b. (\lceil (\text{coin}_1 \neq \text{coin}_2)\langle 1 \rangle = b \rceil * C_{\kappa(b)} c_1. \lceil \text{coin}_1\langle 1 \rangle = c_1 \rceil) \quad (\text{SURE-STR-CONVEX}) \\
\vdash C_\beta b. (\lceil (\text{coin}_1 \neq \text{coin}_2)\langle 1 \rangle = b \rceil * \lceil b = 1 \rceil \Rightarrow C_{\text{Ber}_{\frac{1}{2}}} c_1. \lceil \text{coin}_1\langle 1 \rangle = c_1 \rceil) \quad (\text{C-CONS}) \\
\vdash C_\beta b. (\lceil (\text{coin}_1 \neq \text{coin}_2)\langle 1 \rangle = b \rceil * \lceil b = 1 \rceil \Rightarrow \text{coin}_1\langle 1 \rangle \sim \text{Ber}_{\frac{1}{2}}) \quad (\text{C-UNIT-R})
\end{array}$$

The application of **C-FUSE** allows us to first condition on $\text{coin}_1 \neq \text{coin}_2$, and then the first coin. We can then weaken the case where $b = 0$ and only record that if $b = 1$ then coin_1 is a fair coin.

This takes us through step (85) of Fig. 15. Now the precondition of the if statement is conditional on len and $\text{coin}_1 \neq \text{coin}_2$. Intuitively, we want to evaluate the effects of the if statement in the two possible outcomes and put together the results. This is precisely the purpose of the **C-WP-SWAP** rule, which together with **C-CONS** gives us the derived rule:

$$\begin{array}{c} \text{C-WP-ELIM} \\ \frac{\forall v \in \text{supp}(\mu). P(v) \Vdash \mathbf{wp} \, t \, \{Q(v)\}}{C_\mu v. P(v) \Vdash \mathbf{wp} \, t \, \{C_\mu v. Q(v)\}} \end{array}$$

By applying the rule twice (once on the conditioning on len , and the on the conditioning on $\text{coin}_1 \neq \text{coin}_2$), we can process the if statement case by case, and then combine, under the same conditioning, the postconditions we obtained in each case. The “else” branch is a **skip** (omitted) so it preserves the precondition in the $b = 0$ branch. In the “then” branch we can assume $b = 1$; we apply **WP-RL-ASSIGN** to the assignments and combine the result with the “else” branch by making the overall postcondition of the if statement to be parametric on the value of b (and ℓ).

The last non-obvious step is (86) in Fig. 15, where we show that the conditional postcondition of the if statement implies the loop invariant $P(i + 1)$. Let

$$K(\ell, b) = \begin{cases} \lceil \text{len}\langle 1 \rangle = \ell + 1 \leq i + 1 \rceil * \text{out}[\text{len}]\langle 1 \rangle \sim \text{Ber}_{\frac{1}{2}} & \text{if } b = 1 \\ \lceil \text{len}\langle 1 \rangle = \ell \leq i + 1 \rceil & \text{if } b = 0 \end{cases}$$

then the step is proven as follows:

$$\begin{aligned} & C_\mu \ell. C_\beta b. (\lceil (\text{coin}_1 \neq \text{coin}_2)\langle 1 \rangle = b \rceil * \text{Out}_\ell * K(\ell, b)) \\ & \vdash C_{\mu \otimes \beta}(\ell, b). (\lceil (\text{coin}_1 \neq \text{coin}_2)\langle 1 \rangle = b \rceil * \text{Out}_\ell * K(\ell, b)) \quad (\text{C-ASSOC}) \\ & \vdash C_{\mu \otimes \beta}(\ell, b). (\text{Out}_\ell * K(\ell, b)) \quad (\text{C-CONS}) \\ & \vdash C_{\mu''}(\ell', \ell). \begin{cases} \lceil \text{len}\langle 1 \rangle = \ell' \leq i + 1 \rceil * \text{Out}_{\ell'-1} * \text{out}[\text{len}]\langle 1 \rangle \sim \text{Ber}_{\frac{1}{2}} & \text{if } \ell' = \ell + 1 \\ \lceil \text{len}\langle 1 \rangle = \ell' \leq i + 1 \rceil * \text{Out}_{\ell'} & \text{if } \ell' = \ell \end{cases} \quad (\text{C-TRANSF}) \\ & \vdash C_{\mu'' \circ \pi_1^{-1}} \ell'. (\lceil \text{len}\langle 1 \rangle = \ell' \leq i + 1 \rceil * \text{Out}_{\ell'}) \quad (\text{C-DIST-PROJ}) \\ & \vdash \exists \mu'. C_{\mu'} \ell'. (\lceil \text{len}\langle 1 \rangle = \ell' \leq i + 1 \rceil * \text{Out}_{\ell'}) \end{aligned}$$

The application of **C-TRANSF** uses the function $f(\ell, b) = (\ell + b, \ell)$ to introduce the new ℓ' and then we project away the unused ℓ using the derived **C-DIST-PROJ** (note that the rule applies to $\lceil \cdot \rceil$ assertions and multiple ownership assertions in a separating conjunction thanks to **PROD-SPLIT** and **PROD-UNSPILT**).

G.7 Monte Carlo: BETW_SEQ ≤ BETW

Recall the example sketched in Section 1 where one wants to compare the accuracy of variants of a Monte Carlo algorithm (in Fig. 1) to estimate whether a number x is within the extrema of some set S . Figure 13 reproduces the code here for convenience, with the self-assignments to l and r expanded to their form with a temporary (primed) variable storing the old value of the assigned variable.

The verification task we accomplish in this section is to compare the accuracy of the two Monte Carlo algorithms **BETW_SEQ** and **BETW** (the optimized one).

This goal can be encoded as the judgment:

$$\lceil \text{l}\langle 1 \rangle = \text{r}\langle 1 \rangle = \text{l}\langle 2 \rangle = \text{r}\langle 2 \rangle = 0 \rceil @ \mathbf{p} \vdash \mathbf{wp} \, [1: \text{BETW_SEQ}(x, S), 2: \text{BETW}(x, S)] \, \{ \lceil \text{d}\langle 1 \rangle \leq \text{d}\langle 2 \rangle \rceil \}$$

where \mathbf{p} contains full permissions for all the variables. The judgment states, through the relational lifting, that it is more likely to get a positive answer from BETW than from BETW_SEQ. The challenge is implementing the intuitive relational argument sketched in Section 1, in the presence of very different looping structures.

By **WP-RL-ASSIGN**, it is easy to prove that

$$\llbracket 1\langle 1 \rangle \leq 1\langle 2 \rangle \wedge r\langle 1 \rangle \leq r\langle 2 \rangle \rrbracket @ \mathbf{p} \vdash \mathbf{wp} [1: d := r \& \& 1, 2: d := r \& \& 1] \{ \llbracket d\langle 1 \rangle \leq d\langle 2 \rangle \rrbracket \}$$

Therefore we will focus on proving that the loops produce distributions satisfying $Q = \llbracket 1\langle 1 \rangle \leq 1\langle 2 \rangle \wedge r\langle 1 \rangle \leq r\langle 2 \rangle \rrbracket$.

Now the main obstacle is that we have a single loop at component 2 looping $2N$ times, and two sequentially composed loops in 1, each running N iterations. In a standard coupling-based logic like pRHL, such structural differences are usually bridged by invoking a syntactic transformation (e.g. loop splitting) that is provided by a library of transformations that were proven separately, using meta-reasoning directly on the semantic model, by the designer of the logic. In BLUEBELL we aim at:

- Avoiding resorting to syntactic transformations;
- Avoiding relying on an ad-hoc (incomplete) library of transformations;
- Avoiding having to argue for correctness of transformations semantically.

To achieve this, we formulate the loop-splitting pattern as a *rule* which allows to consider N iterations of component 2 against the first loop of 1, and the rest against the second loop of 1.

WP-LOOP-SPLIT

$$\frac{\begin{array}{c} P_1(N_1) \vdash P_2(0) \\ \forall i < N_1. P_1(i) \vdash \mathbf{wp} [1: t_1, 2: t] \{P_1(i+1)\} \\ \forall j < N_2. P_2(j) \vdash \mathbf{wp} [1: t_2, 2: t] \{P_2(j+1)\} \end{array}}{P_1(0) \vdash \mathbf{wp} [1: (\mathbf{repeat} \ N_1 \ t_1; \mathbf{repeat} \ N_2 \ t_2), 2: \mathbf{repeat} \ (N_1 + N_2) \ t] \{P_2(N_2)\}}$$

Most importantly, such rule is *derivable* from the primitive rules of BLUEBELL, avoiding semantic reasoning all together. Once this rule is proven, it can be used any time need for such pattern arises. Before showing how this rule is derivable, which we do in Lemma G.1, let us show how to use it to close our example.

We want to apply **WP-LOOP-SPLIT** with $N_1 = N_2 = N$, t_1 as the body of the loop of BelowMax, t_2 as the body of the loop of AboveMin, and t as the body of the loop of BETW. We define the two loop invariants as follows:

$$P_1(i) \triangleq \llbracket r\langle 1 \rangle \leq r\langle 2 \rangle \wedge 1\langle 1 \rangle = 0 \leq 1\langle 2 \rangle \rrbracket \quad P_2(j) \triangleq \llbracket r\langle 1 \rangle \leq r\langle 2 \rangle \wedge 1\langle 1 \rangle \leq 1\langle 2 \rangle \rrbracket$$

Note that they both ignore the iteration number. Clearly we have:

$$P_0 \vdash P_1(0) \quad P_1(N) \vdash P_2(0) \quad P_2(N) \vdash Q$$

By applying **WP-LOOP-SPLIT** we reduce the goal to the triples:

$$\begin{array}{l} \llbracket r\langle 1 \rangle \leq r\langle 2 \rangle \wedge 1\langle 1 \rangle = 0 \leq 1\langle 2 \rangle \rrbracket \vdash \mathbf{wp} [1: t_1, 2: t] \{ \llbracket r\langle 1 \rangle \leq r\langle 2 \rangle \wedge 1\langle 1 \rangle = 0 \leq 1\langle 2 \rangle \rrbracket \} \\ \llbracket r\langle 1 \rangle \leq r\langle 2 \rangle \wedge 1\langle 1 \rangle \leq 1\langle 2 \rangle \rrbracket \vdash \mathbf{wp} [1: t_2, 2: t] \{ \llbracket r\langle 1 \rangle \leq r\langle 2 \rangle \wedge 1\langle 1 \rangle \leq 1\langle 2 \rangle \rrbracket \} \end{array}$$

which are easy to obtain by replicating the standard coupling-based reasoning steps, using **COUPLING** and **WP-RL-ASSIGN**.

As promised, we now prove **WP-LOOP-SPLIT** is derivable.

LEMMA G.1. *Section 5.6 is sound.*

PROOF. Assume:

$$P_1(N_1) \vdash P_2(0) \quad (87)$$

$$\forall i < N_1. P_1(i) \vdash \mathbf{wp} [1: t_1, 2: t] \{P_1(i+1)\} \quad (88)$$

$$\forall j < N_2. P_2(j) \vdash \mathbf{wp} [1: t_2, 2: t] \{P_2(j+1)\} \quad (89)$$

We want to show:

$$P_1(0) \vdash \mathbf{wp} [1: (\mathbf{repeat} N_1 t_1; \mathbf{repeat} N_2 t_2), 2: \mathbf{repeat} (N_1 + N_2) t] \{P_2(N_2)\}$$

First, by using **WP-NEST** and **WP-SEQ**, we can reduce the goal to:

$$P_1(0) \vdash \mathbf{wp} [2: \mathbf{repeat} (N_1 + N_2) t] \{ \mathbf{wp} [1: \mathbf{repeat} N_1 t_1] \{ \mathbf{wp} [1: \mathbf{repeat} N_2 t_2] \{P_2(N_2)\} \} \}$$

Now define:

$$P(k) = \begin{cases} \mathbf{wp} [1: \mathbf{repeat} k t_1] \{P_1(k)\} & \text{if } k \leq N_1 \\ \mathbf{wp} [1: \mathbf{repeat} N_1 t_1] \{ \mathbf{wp} [1: \mathbf{repeat} (k - N_1) t_2] \{P_2(k - N_1)\} \} & \text{if } k > N_1 \end{cases}$$

We have:

$$P_1(0) \vdash P(0) \quad (90)$$

$$P(N_1 + N_2) \vdash \mathbf{wp} [1: \mathbf{repeat} N_1 t_1] \{ \mathbf{wp} [1: \mathbf{repeat} N_2 t_2] \{P_2(N_2)\} \} \quad (91)$$

Entailment (90) holds by **WP-LOOP-0**, and (91) holds by definition. Therefore, using **WP-CONS** we reduced the goal to

$$P(0) \vdash \mathbf{wp} [2: \mathbf{repeat} (N_1 + N_2) t] \{P(N_1 + N_2)\}$$

which we can make progress on using **WP-LOOP**. We are left with proving:

$$\forall k < N_1 + N_2. P(k) \vdash \mathbf{wp} [2: t] \{P(k+1)\}$$

We distinguish three cases:

– **Case** $k < N_1$. By unfolding the definition of P we obtain:

$$\mathbf{wp} [1: \mathbf{repeat} k t_1] \{P_1(k)\} \vdash \mathbf{wp} [2: t] \{ \mathbf{wp} [1: \mathbf{repeat} (k+1) t_1] \{P_1(k+1)\} \}$$

Using **WP-LOOP-UNF** on the inner WP we obtain:

$$\mathbf{wp} [1: \mathbf{repeat} k t_1] \{P_1(k)\} \vdash \mathbf{wp} [2: t] \{ \mathbf{wp} [1: \mathbf{repeat} (k) t_1] \{ \mathbf{wp} [1: t_1] \{P_1(k+1)\} \} \}$$

By **WP-NEST** we can swap the two topmost WPs:

$$\mathbf{wp} [1: \mathbf{repeat} k t_1] \{P_1(k)\} \vdash \mathbf{wp} [1: \mathbf{repeat} k t_1] \{ \mathbf{wp} [2: t] \{ \mathbf{wp} [1: t_1] \{P_1(k+1)\} \} \}$$

Finally, by **WP-CONS** we can eliminate the topmost WP from both sides:

$$P_1(k) \vdash \mathbf{wp} [2: t] \{ \mathbf{wp} [1: t_1] \{P_1(k+1)\} \}$$

which by **WP-NEST** is our assumption (88) with $i = k$.

– **Case** $k = N_1$. By unfolding the definition of P we obtain:

$$\mathbf{wp} [1: \mathbf{repeat} N_1 t_1] \{P_1(N_1)\} \vdash \mathbf{wp} [2: t] \{ \mathbf{wp} [1: \mathbf{repeat} N_1 t_1] \{ \mathbf{wp} [1: \mathbf{repeat} 1 t_2] \{P_2(0)\} \} \}$$

By a trivial application of **WP-LOOP** we have $\mathbf{wp} [1: t] \{Q\} \vdash \mathbf{wp} [1: \mathbf{repeat} 1 t] \{Q\}$, so we can simplify the innermost WP to:

$$\mathbf{wp} [1: \mathbf{repeat} N_1 t_1] \{P_1(N_1)\} \vdash \mathbf{wp} [2: t] \{ \mathbf{wp} [1: \mathbf{repeat} N_1 t_1] \{ \mathbf{wp} [1: t_2] \{P_2(1)\} \} \}$$

Then by **WP-NEST** we can swap the topmost WPs:

$$\mathbf{wp} [1: \mathbf{repeat} N_1 t_1] \{P_1(N_1)\} \vdash \mathbf{wp} [1: \mathbf{repeat} N_1 t_1] \{ \mathbf{wp} [2: t] \{ \mathbf{wp} [1: t_2] \{P_2(1)\} \} \}$$

By **WP-CONS** we can eliminate the topmost WP from both sides:

$$P_1(N_1) \vdash \text{wp} [2:t] \{ \text{wp} [1:t_2] \{P_2(1)\} \}$$

Using assumption (91) we can reduce this to:

$$P_2(0) \vdash \text{wp} [2:t] \{ \text{wp} [1:t_2] \{P_2(1)\} \}$$

which by **WP-NEST** is our assumption (89) with $j = 0$.

– **Case** $k > N_1$. By unfolding the definition of P we obtain:

$$\begin{aligned} & \text{wp} [1:\text{repeat } N_1 t_1] \{ \text{wp} [1:\text{repeat } (k - N_1) t_2] \{P_2(k - N_1)\} \} \\ & \vdash \text{wp} [2:t] \{ \text{wp} [1:\text{repeat } N_1 t_1] \{ \text{wp} [1:\text{repeat } (k - N_1 + 1) t_2] \{P_2(k - N_1 + 1)\} \} \} \end{aligned}$$

Using **WP-LOOP-UNF** on the inner WP we obtain:

$$\begin{aligned} & \text{wp} [1:\text{repeat } N_1 t_1] \{ \text{wp} [1:\text{repeat } (k - N_1) t_2] \{P_2(k - N_1)\} \} \\ & \vdash \text{wp} [2:t] \{ \text{wp} [1:\text{repeat } N_1 t_1] \{ \text{wp} [1:\text{repeat } (k - N_1) t_2] \{ \text{wp} [1:t_2] \{P_2(k - N_1 + 1)\} \} \} \} \end{aligned}$$

By **WP-NEST** we can push the topmost WP inside:

$$\begin{aligned} & \text{wp} [1:\text{repeat } N_1 t_1] \{ \text{wp} [1:\text{repeat } (k - N_1) t_2] \{P_2(k - N_1)\} \} \\ & \vdash \text{wp} [1:\text{repeat } N_1 t_1] \{ \text{wp} [1:\text{repeat } (k - N_1) t_2] \{ \text{wp} [2:t] \{ \text{wp} [1:t_2] \{P_2(k - N_1 + 1)\} \} \} \} \end{aligned}$$

Finally, by **WP-CONS** we can eliminate the topmost WPs from both sides:

$$P_2(k - N_1) \vdash \text{wp} [2:t] \{ \text{wp} [1:t_2] \{P_2(k - N_1 + 1)\} \}$$

which by **WP-NEST** is our assumption (89) with $j = k - N_1$. \square

G.8 Monte Carlo: Equivalence Between BETW_MIX and BETW_SEQ

Figure 13 shows another way in which one can approximately compute the “between” function: BETW_MIX. In this example we want to prove the equivalence between BETW_MIX and BETW_SEQ. Again, the main obstacle to overcome in the proof is that the structure of the two programs is very different. BETW_SEQ has two loops of N iterations, with one sample per iteration. BETW_MIX has a single loop of N iterations, but it samples twice per iteration. Note that the equivalence cannot be understood as a generic program transformation: the order in which the samples are taken in the two programs is drastically different; they are only equivalent because the calculations done on each of these independent samples are independent from one another.

Intuitively, we want to produce a proof that aligns each iteration of the first loop of BETW_SEQ with half of each iteration of BETW_MIX, and each iteration of the second loop of BETW_SEQ with the second half of each iteration of BETW_MIX. In the same vein as the previous example, we want to formalize the proof pattern as a rule that aligns the loops as desired, prove the rule is derivable, and apply it to the example. A rule encoding the above pattern is the following:

WP-LOOP-MIX

$$\frac{\forall i < N. P_1(i) \vdash \text{wp} [1:t_1, 2:t'_1] \{P_1(i + 1)\} \quad \forall i < N. P_2(i) \vdash \text{wp} [1:t_2, 2:t'_2] \{P_2(i + 1)\}}{P_1(0) * P_2(0) \vdash \text{wp} [1:(\text{repeat } N t_1; \text{repeat } N t_2), 2:\text{repeat } N (t'_1; t'_2)] \{P_1(N) * P_2(N)\}}$$

Before showing how this rule is derivable, which we do in Lemma G.2, let us show how to use it to close our example.

We want to prove the goal:

$$\left(\begin{array}{l} \lceil r\langle 1 \rangle = 1\langle 1 \rangle = 0 \rceil \\ \lceil r\langle 2 \rangle = 1\langle 2 \rangle = 0 \rceil \end{array} \right) @P \vdash \text{wp} \left[\begin{array}{l} 1: \text{repeat } N t_r; \text{repeat } N t_1 \\ 2: \text{repeat } N (t_r; t_1) \end{array} \right] \left\{ \begin{array}{l} \lceil r\langle 1 \rangle = r\langle 2 \rangle \rceil \\ \lceil 1\langle 1 \rangle = 1\langle 2 \rangle \rceil \end{array} \right\}$$

where \mathbf{p} has full permissions for all the relevant variables, t_r is the body of the loop of BelowMax, and t_1 is the body of the loop of AboveMin.

As a first manipulation, we use **RL-MERGE** in the postcondition, and Section 2.3 (via **SURE-DIRAC**) to the precondition, to obtain:

$$\left(\begin{array}{l} \llbracket r\langle 1 \rangle = r\langle 2 \rangle \rrbracket @ \mathbf{p}_r * \\ \llbracket l\langle 1 \rangle = l\langle 2 \rangle \rrbracket @ \mathbf{p}_1 \end{array} \right) \vdash \mathbf{wp} \left[\begin{array}{l} \mathbf{1: repeat } N \ t_r ; \mathbf{repeat } N \ t_1 \\ \mathbf{2: repeat } N \ (t_r ; t_1) \end{array} \right] \left\{ \left(\begin{array}{l} \llbracket r\langle 1 \rangle = r\langle 2 \rangle \rrbracket @ \mathbf{p}_r * \\ \llbracket l\langle 1 \rangle = l\langle 2 \rangle \rrbracket @ \mathbf{p}_1 \end{array} \right) \right\}$$

where $\mathbf{p}_r = [r\langle 1 \rangle: 1, r\langle 2 \rangle: 1, q\langle 1 \rangle: 1, q\langle 2 \rangle: 1]$, and $\mathbf{p}_1 = [l\langle 1 \rangle: 1, l\langle 2 \rangle: 1, p\langle 1 \rangle: 1, p\langle 2 \rangle: 1]$. Then **WP-LOOP-MIX** applies and we are left with the two triples

$$\begin{aligned} \llbracket r\langle 1 \rangle = r\langle 2 \rangle \rrbracket @ \mathbf{p}_r &\vdash \mathbf{wp} [\mathbf{1: } t_r, \mathbf{2: } t_r] \{ \llbracket r\langle 1 \rangle = r\langle 2 \rangle \rrbracket @ \mathbf{p}_r \} \\ \llbracket l\langle 1 \rangle = l\langle 2 \rangle \rrbracket @ \mathbf{p}_1 &\vdash \mathbf{wp} [\mathbf{1: } t_1, \mathbf{2: } t_1] \{ \llbracket l\langle 1 \rangle = l\langle 2 \rangle \rrbracket @ \mathbf{p}_1 \} \end{aligned}$$

which are trivially proved using a standard coupling argument.

As promised, we now prove **WP-LOOP-MIX** is derivable, concluding the example.

LEMMA G.2. *Section 5.6 is sound.*

PROOF. Assume:

$$\forall i < N. P_1(i) \vdash \mathbf{wp} [\mathbf{1: } t_1, \mathbf{2: } t'_1] \{P_1(i+1)\} \quad (92)$$

$$\forall i < N. P_2(i) \vdash \mathbf{wp} [\mathbf{1: } t_2, \mathbf{2: } t'_2] \{P_2(i+1)\} \quad (93)$$

Our goal is to prove:

$$P_1(0) * P_2(0) \vdash \mathbf{wp} [\mathbf{1: (repeat } N \ t_1 ; \mathbf{repeat } N \ t_2), \mathbf{2: repeat } N \ (t'_1 ; t'_2)] \{P_1(N) * P_2(N)\}$$

We first massage the goal to split the sequential composition at **1**. By **WP-SEQ** and **WP-NEST** we obtain

$$\begin{aligned} P_1(0) * P_2(0) &\vdash \mathbf{wp} [\mathbf{2: repeat } N \ (t'_1 ; t'_2)] \\ &\quad \vdash \{ \mathbf{wp} [\mathbf{1: repeat } N \ t_1] \{ \mathbf{wp} [\mathbf{1: repeat } N \ t_2] \{P_1(N) * P_2(N)\} \} \} \end{aligned}$$

Now by applying **WP-FRAME** in the postcondition (twice) we obtain

$$P_1(0) * P_2(0) \vdash \mathbf{wp} [\mathbf{2: repeat } N \ (t'_1 ; t'_2)] \left\{ \left(\begin{array}{l} \mathbf{wp} [\mathbf{1: repeat } N \ t_1] \{P_1(N)\} * \\ \mathbf{wp} [\mathbf{1: repeat } N \ t_2] \{P_2(N)\} \end{array} \right) \right\} \quad (94)$$

Define

$$P(i) \triangleq Q_1(i) * Q_2(i) \quad Q_1(i) \triangleq \mathbf{wp} [\mathbf{1: repeat } i \ t_1] \{P_1(i)\} \quad Q_2(i) \triangleq \mathbf{wp} [\mathbf{1: repeat } i \ t_2] \{P_2(i)\}$$

Clearly we have $P_1(0) * P_2(0) \vdash P(0)$ (by **WP-LOOP-0**) and $P(N)$ coincides with the postcondition of our goal (94), which is now rewritten to:

$$P(0) \vdash \mathbf{wp} [\mathbf{2: repeat } N \ (t'_1 ; t'_2)] \{P(N)\}$$

Now we can apply **WP-LOOP** with invariant P and reduce the goal to the triples:

$$\forall i < N. Q_1(i) * Q_2(i) \vdash \mathbf{wp} [\mathbf{2: } t'_1 ; t'_2] \{Q_1(i+1) * Q_2(i+1)\}$$

By **WP-SEQ** and **WP-FRAME** we can reduce the goal to

$$Q_1(i) * Q_2(i) \vdash \mathbf{wp} [\mathbf{2: } t'_1] \{Q_1(i+1)\} * \mathbf{wp} [\mathbf{2: } t'_2] \{Q_2(i+1)\}$$

which we can prove by showing the two triples:

$$Q_1(i) \vdash \mathbf{wp} [\mathbf{2: } t'_1] \{Q_1(i+1)\} \quad Q_2(i) \vdash \mathbf{wp} [\mathbf{2: } t'_2] \{Q_2(i+1)\}$$

We focus on the former as the latter can be dealt with symmetrically. By unfolding Q_1 we obtain:

$$\text{wp } [1: \text{repeat } i \ t_1] \{P_1(i)\} \vdash \text{wp } [2: t'_1] \{ \text{wp } [1: \text{repeat } (i+1) \ t_1] \{P_1(i+1)\} \}.$$

We then apply **WP-LOOP-UNF** to the innermost WP and **WP-NEST** to swap the two WPs in the conclusion:

$$\text{wp } [1: \text{repeat } i \ t_1] \{P_1(i)\} \vdash \text{wp } [1: \text{repeat } i \ t_1] \{ \text{wp } [2: t'_1, 1: t_1] \{P_1(i+1)\} \}.$$

Finally, by **WP-CONS** we can eliminate the topmost WPs on both sides and reduce the goal to assumption (92). \square

G.9 Randomized Cache Management

Cache is a part of computer memory fast to access but limited in its size. Cache management algorithms determine which elements are to keep or evict upon a sequence of requests. If the computer knows the entire sequence of requests, then its best strategy is to always evict the element that is next requested furthest in the future. We call this strategy the *offline optimal algorithm*. However, a realistic cache algorithm needs to make decisions online, which make it impossible to consider requests in the future. While for some applications deterministic cache algorithms, such as First-In-First-Out (FIFO), Last-In-First-Out (LIFO), and Least Recently Used (LRU), perform relatively well, they suffer when the sequence of requests are produced adversarially to maximize their misses. It is proved that, for a cache that holds k items, no deterministic cache algorithm is $< k$ -competitive; that means for any deterministic cache algorithm, there exists some sequence on which the algorithm produces at least k times misses of the offline optimal algorithm's number of misses.

Randomization could help a lot in this scenario of adversarial requests. The *Marker algorithm*, a randomized algorithm also known as 1-bit LRU algorithm, is shown to be $\log(k)$ -competitive to the offline optimal. While the algorithm is relatively intuitive, its analysis involves many parts. Though we do not have an end-to-end formal proof of that analysis, we can verify some important components in our logic.

The Marker algorithm attaches each slot of its cache with an 1-bit marker. Initially, for any $1 \leq i \leq k$, the mark $\text{cache}[i][\text{mark}]$ is set to 0. Then it operates in phases. At the beginning of each phase, the markers are all set to 0. Every time it encounters a new request that it has not seen in the current phase, either the requested element is already in the cache and then it simply changes that slot's mark to 1, or the requested element is not in the cache, then it replaces the item on a randomly chosen 0-marked slot with the request and marks that slot to 1. Thus, after processing k distinct requests in a phase, all the slots in the cache are marked with 1. The algorithm then reset all these marks to 0 and start a new phase.

We show the details of the implementation of each phase in Fig. 16. (The part resetting all marks to 0 and starts a new phase is not modelled.) A crucial step in the implementation is to sample the eviction index ev uniformly randomly from all index marked to 0. There are many ways to implement that. We use a memory-efficient reservoir sampling algorithm [Vitter 1985], which can also be used in many other applications. Throughout this example, in both the code and the proof, we use $[a, b]$ to denote the set of integers that are at least a and at most b .

G.9.1 Reservoir Sampling. The first component that we verify is the reservoir sampling. We want to show that for any input cache, the output ev is distributed uniformly among indices $1 \leq l \leq k$ with $\text{cache}[l][\text{mark}] = 0$. We state that goal in our logic as

$$\text{VS. } [l \in S \Leftrightarrow \text{cache}[l][\text{mark}] = 1]$$

$$\models \text{wp } 1 : \text{Reservoir-Samp}(k, \text{cache}) \{ [l \in S \Leftrightarrow \text{cache}[l][\text{mark}] = 1] * ev \sim \text{Unif}_{[1, k] \setminus S} \}$$


```

def Marker(seq, N, cache):
    cost := 0
    i := 1
    repeat N:
        hit := False
        j := 1
        repeat k:
            if cache[j] == seq[i]:
                cache[j] := (seq[i], 1)
                hit := True
            j := j + 1
        if hit:
            skip
        else:
            ev ≈ Reservoir_Samp(k, cache)
            cache[ev] := (seq[i], 1)
            cost := cost + 1
        i := i + 1

def Reservoir_Samp(k, cache):
    c := 0
    j := 0
    ev := None
    repeat k:
        c := c + 1
        if cache[j][mark] = 0 then
            j := j + 1
            x ≈ unifInt [1, j]
            if x ≤ 1:
                ev := c

```

Fig. 16. The Marker algorithm.

The condition $[l \in S \Leftrightarrow \text{cache}[l][\text{mark}] = 1]$ makes sure that S contains exactly the cache indices marked as 0.

The program uses c to track the number of slots that we iterate over and j to count the number of cache indices with mark 0. It initializes both c and j to 0, and ev to a non-integer value `None`. The program then iterates over the entries $\text{cache}[c]$ for $1 \leq c \leq k$ and in each iteration update ev and j to maintain that the loop invariants that 1) ev is distributed uniformly among indices we have iterated over that are marked 0, i.e., $ev \sim \text{Unif}_{[1,c] \setminus S}$ (the uniform distribution over empty set does not make sense, so we arbitrarily interpret Unif_\emptyset as the Dirac distribution over the value `None`); 2) the number of element in $\text{Unif}_{[1,c] \setminus S}$ is j . Specifically, in the c^{th} iteration, if $\text{cache}[c] = 1$, we just increase c by 1 and enter the next iteration. If $\text{cache}[c] = 0$, then we increase both c , j by 1 and sample an integer x uniformly from the interval $[1, j]$. If $x = 1$, which happens with probability $\frac{1}{j}$, then we update ev to the current index c ; if $x \neq 1$, which happens with $\frac{j-1}{j}$ of the case, we keep the previous ev , which has equal probability to be any one of the $j - 1$ elements from $[1, c - 1] \setminus S$. Thus, the distribution of ev after this iteration is a convex combination of these two branches, which we can show to be $\text{Unif}_{[1,c] \setminus S}$, thus reestablishing the loop invariants.

We outline the proof in 17. We omit `ownx` and permission 1 on all variables get assigned to in all assertions. Also, because we need to include $[l \in S \Leftrightarrow \text{cache}[l][\text{mark}] = 1]$ in all the assertions, we abbreviate it into ϕ to minimize the visual overhead.

Define $P(c)$ to be

$$\phi * ev \sim \text{Unif}_{[1,c] \setminus S} * [| [1, c] \setminus S | = j]$$

At the first few steps, by **WP-ASSIGN**, we get

$$\phi * [c = 0] * [j = 0] * [ev = \text{None}]$$

```

{[l ∈ S ⇔ cache[l][mark] = 1]}
c = 0
j = 0
ev := None
{φ * [c = 0] * [j = 0] * [ev = None]}
repeat k:
  {P(c)}
  {φ * ev ~ Unif[1,c] \ S * |[1, c] \ S| = j]}
  c := c + 1
  {φ * ev ~ Unif[1,c-1] \ S * |[1, c-1] \ S| = j]}
  if cache[c][mark] = 0 then
    j := j+1
    {[cache[c][mark] = 0] * φ * ev ~ Unif[1,c-1] \ S * |[1, c-1] \ S| = j-1]}
    x := unifInt [1, j]
    {[cache[c][mark] = 0] * φ * ev ~ Unif[1,c-1] \ S * |[1, c-1] \ S| = j-1] * x ~ Unif[1,j]}}
    if x = 1:
      ev := c
      {
        CUnif[1,j] w. (
          [w ≠ 1] ⇒ CUnif[1,c-1] \ S v. [ev = v]
          [w = 1] ⇒ [ev = c]
        ) * (
          [cache[c][mark] = 0]
          * φ * |[1, c-1] \ S| = j-1
        )
        {
          [cache[c][mark] = 0] * φ * ev ~ Unif[1,c] \ S * |[1, c] \ S| = j
        }
        {
          [cache[c][mark] = 0] ⇒ φ * ev ~ Unif[1,c] \ S * |[1, c] \ S| = j
          [cache[c][mark] = 1] ⇒ φ * ev ~ Unif[1,c] \ S * |[1, c] \ S| = |[1, c-1] \ S| = j
        }
        {φ * ev ~ Unif[1,c] \ S * |[1, c] \ S| = j}
        {φ * ev ~ Unif[1,k] \ S * |[1, k] \ S| = j}
      }
  }

```

(96)

Fig. 17. Reservoir Sampling

before entering the first iteration. This propositionally implies that $[1, c] \setminus S = \emptyset$, and by **C-UNIT-L** and **C-UNIT-R**,

$$[ev = \text{None}] \vdash ev \sim \delta_{\text{None}}$$

and thus we have $P(0)$.

We then want to use **WP-IF-PRIM** to prove (96). The $[cache[c][mark] = 1]$ branch is trivial. For the $[cache[c][mark] = 0]$ branch, we first apply **WP-ASSIGN** and **WP-SAMP** for the assignment and the sampling. We then apply **WP-IF-UNARY** and **WP-ASSIGN** for the **if** $x = 1$ branching. The most crucial step is (95), which we prove in the follow. For visual presentation, we abbreviate $[cache[c][mark] = 0] * \phi$ as ψ .

$$\begin{aligned}
& C_{\text{Unif}_{[1,j]}} w. \left(\begin{array}{l} [w \neq 1] \Rightarrow C_{\text{Unif}_{[1,c-1] \setminus S}} v. [ev = v] \\ [w = 1] \Rightarrow [ev = c] \end{array} \right) * \psi * |[1, c-1] \setminus S| = j-1 \\
& \vdash C_{\text{Unif}_{[1,j]}} w. \left(\begin{array}{l} [w \neq 1] \Rightarrow C_{\text{Unif}_{[1,c-1] \setminus S}} v. [ev = v] \\ [w = 1] \Rightarrow C_{\text{Unif}_{[1,c-1] \setminus S}} v. [ev = c] \end{array} \right) * \psi * |[1, c-1] \setminus S| = j-1 \\
& \quad \text{(C-TRUE, C-FRAME)} \\
& \vdash C_{\text{Unif}_{[1,j]}} w. C_{\text{Unif}_{[1,c-1] \setminus S}} v. [ev = \text{if } w = 1 \text{ then } c \text{ else } v] * \psi * |[1, c-1] \setminus S| = j-1 \quad \text{(C-CONS)} \\
& \vdash C_{\text{Unif}_{[1,j]} \otimes \text{Unif}_{[1,c-1] \setminus S}} (w, v). [ev = \text{if } w = 1 \text{ then } c \text{ else } v] * \psi * |[1, c-1] \setminus S| = j-1 \quad \text{(C-FUSE)}
\end{aligned}$$

Because $[|[1, c-1] \setminus S| = j-1]$, there exists some bijection h between $[1, c-1] \setminus S$ and $[1, j-1]$. For $(x, y) \in [1, j-1] \times (([1, c-1] \setminus S) \cup \{c\})$, we define

$$f(x, y) = \text{if } y = c \text{ then } 1 \text{ else } x + 1$$

$$g(x, y) = \text{if } y = c \text{ then } h^{-1}(x) \text{ else } y$$

Pure reasoning yields that the map $\langle f, g \rangle : (w, v) \mapsto (f(w, v), g(w, v))$ is a bijection between $[1, j-1] \times (([1, c-1]) \setminus S) \cup \{c\}$ and $[1, j] \times ([1, c-1] \setminus S)$. Furthermore, for any (x, y) ,

$$\text{Unif}_{[1,j]} \otimes \text{Unif}_{[1,c-1] \setminus S}(f(x), g(y)) = \frac{1}{j \cdot (j-1)} = \text{Unif}_{[1,j-1]} \otimes \text{Unif}_{[1,(c-1) \setminus S] \cup \{c\}}(x, y),$$

Thus, we can apply **C-TRANSF** with the bijection $\langle f, g \rangle$ to derive that

$$\begin{aligned} & \mathcal{C}_{\text{Unif}_{[1,j]} \otimes \text{Unif}_{[1,c-1] \setminus S}}(w, v). [\text{ev} = \text{if } w = 1 \text{ then } c \text{ else } v] * \psi * \llbracket [1, c-1] \setminus S \rrbracket = j-1 \\ & \vdash \mathcal{C}_{\text{Unif}_{[1,j-1]} \otimes \text{Unif}_{([1,c-1] \setminus S) \cup \{c\}}}(x, y). \left(\llbracket \text{ev} = \text{if } f(x, y) = 1 \text{ then } c \text{ else } g(x, y) \rrbracket \right) \\ & \quad * \psi * \llbracket [1, c-1] \setminus S \rrbracket = j-1. \end{aligned}$$

Propositional reasoning gives the result that

$$\begin{aligned} & \text{if } f(x, y) = 1 \text{ then } c \text{ else } g(x, y) \\ & = \text{if } y = c \text{ then } c \text{ else } (\text{if } y = c \text{ then } h^{-1}(x) \text{ else } y) \\ & = y. \end{aligned}$$

Thus, we can simplify the assertion above into

$$\mathcal{C}_{\text{Unif}_{[1,j-1]} \otimes \text{Unif}_{([1,c-1] \setminus S) \cup \{c\}}}(x, y). [\text{ev} = y] * \psi * \llbracket [1, c-1] \setminus S \rrbracket = j-1.$$

Now recall that ψ is $\llbracket \text{cache}[c][\text{mark}] = 0 \rrbracket * \llbracket l \in S \Leftrightarrow \text{cache}[l][\text{mark}] = 1 \rrbracket$, which implies that the set $([1, c-1] \setminus S) \cup \{c\}$ is the equivalent to $([1, c] \setminus S)$. Thus, we can further simplify the assertion above into

$$\mathcal{C}_{\text{Unif}_{[1,j-1]} \otimes \text{Unif}_{[1,c] \setminus S}}(x, y). [\text{ev} = y] * \psi * \llbracket [1, c] \setminus S \rrbracket = j,$$

Then, we apply **C-SURE-PROJ** to project out the unused part of the distribution under conditioning:

$$\mathcal{C}_{\text{Unif}_{[1,c] \setminus S}}(x, y). \left(\llbracket \text{ev} = y \rrbracket * \llbracket \text{cache}[c][\text{mark}] = 0 \rrbracket \right) * \llbracket l \in S \Leftrightarrow \text{cache}[l][\text{mark}] = 1 \rrbracket * \llbracket [1, c] \setminus S \rrbracket = j.$$

Applying **SURE-STR-CONVEX**, we can pull out almost sure assertions under the conditioning modality and get

$$\left(\mathcal{C}_{\text{Unif}_{[1,c] \setminus S}}(x, y). [\text{ev} = y] \right) * \left(\llbracket \text{cache}[c][\text{mark}] = 0 \rrbracket * \llbracket l \in S \Leftrightarrow \text{cache}[l][\text{mark}] = 1 \rrbracket * \llbracket [1, c] \setminus S \rrbracket = j \rrbracket \right).$$

Last, we apply **C-UNIT-L** to obtain (95), and (96) follows from **WP-IF-UNARY**.

By simplifying (96), we close the loop invariant $P(c)$. Finally, applying the loop rule **WP-LOOP**, we get

$$\llbracket l \in S \Leftrightarrow \text{cache}[l][\text{mark}] = 1 \rrbracket * \text{ev} \sim \text{Unif}_{[1,c] \setminus S} * \llbracket [1, c] \setminus S \rrbracket = j.$$

G.9.2 Marker algorithm within a phase. When reasoning about Marker's algorithm, the pen-and-paper proof uses a reduction-flavored technique. Given the actual sequence of requests seq , we compute a modified sequence worseseq by moving requests that are not in the initial cache, a.k.a., the clean request, ahead of those that are in the cache, a.k.a., the dirty requests. Each clean request incurs cost 1 wherever it appears, and dirty request incurs cost 1 if the cache slot holding its value has been evicted in the current phase and incur cost 0 otherwise. Thus, the clean requests in seq and worseseq incur the same cost, and the dirty requests in worseseq incur at least as much cost as in seq . This is an inherently relational argument. The rest part of the pen-and-paper analysis uses a lot of unary reasoning to upper bound the cost of Marker on worseseq , and concludes that it also upper bounds the cost of processing seq . Though we are not able to prove every step in this

analysis formally, it suggests the need to combine unary and n-ary reasoning. In the following, we verify an important component of the unary analysis: when Marker operates on M clean requests in the beginning of a phase, the M distinct clean requests incurs exactly cost m and the evicted slots are chosen uniformly.

At the beginning of the phase, it holds that

$$\forall 1 \leq l \leq k. [\text{cache}[l][\text{mark}] = 0].$$

We also assume that cost is c and cache is h in the beginning of the phase. To express that the Marker is handling requests $\text{seq}[i]$ that are not currently in the cache, we assert that

$$\forall 1 \leq n \leq M. \forall 1 \leq l \leq k. [\text{seq}[n] \neq h[l][\text{value}]].$$

The cost increases by exactly M only if these M clean requests are distinct, so we also require

$$\forall 1 \leq n' < n \leq M. [\text{seq}[n] \neq \text{seq}[n']]$$

We combine these into the precondition and outline the proof at Figure 18. The assertions throughout should be conjuncted with own_x with \wedge and we omit it for visual simplicity. For the loop, we use the loop invariant

$$P(i) := C_{\kappa(i-1)} S. \left(\begin{aligned} & [\text{cost} = c + i - 1] * (\forall i \leq n \leq M. \forall 1 \leq l \leq k. [\text{seq}[n] \neq h[l][\text{value}]]) \\ & * (\forall l. [l \in S \Leftrightarrow \text{cache}[l][\text{mark}] = 1 \Leftrightarrow \text{cache}[l][\text{value}] \neq h[l][\text{value}]]) \\ & * (\forall 1 \leq n' < n \leq M. [\text{seq}[n] \neq \text{seq}[n']]) \end{aligned} \right)$$

where the kernel κ is defined such that for any $1 \leq i \leq m$, $\kappa(i) = \text{Unif}_{\{S \subseteq [1,k] \mid \text{size}(S)=i\}}$ – so $\kappa(i-1) = \text{Unif}_{\{S \subseteq [1,k] \mid \text{size}(S)=i-1\}}$. And we write $[A \Leftrightarrow B \Leftrightarrow C]$ as shorthand for $[A \Leftrightarrow B] * [B \Leftrightarrow C]$.

Starting from $P(i)$, we first apply **WP-ASSIGN** twice and then enter the inner loop with

$$Q(j) = C_{\kappa(i-1)} S. \left(\begin{aligned} & [\text{cost} = c + i - 1] * (\forall 1 \leq n \leq M. \forall 1 \leq l \leq k. [\text{seq}[n] \neq h[l][\text{value}]]) \\ & * (\forall l. [l \in S \Leftrightarrow \text{cache}[l][\text{mark}] = 1 \Leftrightarrow \text{cache}[l][\text{value}] \neq h[l][\text{value}]]) \\ & * [\text{hit} = 0] * (\forall 1 \leq n' < n \leq M. [\text{seq}[n] \neq \text{seq}[n']]) \end{aligned} \right)$$

To reestablish $Q(j)$ in (98), we apply **C-WP-SWAP** to reason from fixed S . Then we use **WP-IF-PRIM**: the condition $\forall 1 \leq n \leq M. \forall 1 \leq l \leq k. [\text{seq}[n] \neq h[l][\text{value}]]$ implies that we do not enter **if** block and the assertion $Q(j)$ still holds afterwards. Last, we apply **WP-ASSIGN** to derive that $Q(j)$ holds after the assignment $j := j+1$. Since $Q(j) = Q(j+1)$, $Q(j+1)$ holds as well. Thus by **WP-LOOP** we have (99).

The step (100) is derived using **C-WP-SWAP** and the specs of Reservoir-Samp. For the step (101), we apply **C-UNIT-R** to rewrite $\text{ev} \sim \text{Unif}_{[1,k] \setminus S}$ as $C_{\text{Unif}_{[1,k] \setminus S}} u. [\text{ev} = u]$ and we use **C-FRAME** to pull $C_{\text{Unif}_{[1,k] \setminus S}} u.$ ahead. In the next step, using **WP-ASSIGN**, we get

$$C_{\kappa(i-1)} S. C_{\text{Unif}_{[1,k] \setminus S}} u. \left(\begin{aligned} & [\text{cost} = c + i - 1] * (\forall i < n \leq M. \forall 1 \leq l \leq k. [\text{seq}[n] \neq h[l][\text{value}]]) \\ & * \left(\begin{aligned} & \forall l. [l \in S \Leftrightarrow \text{cache}[l][\text{mark}] = 1] \\ & * [l \in S \wedge l \neq \text{ev} \Rightarrow \text{cache}[l][\text{value}] \neq h[l][\text{value}]] \\ & * [\text{cache}[l][\text{value}] \neq h[l][\text{value}] \wedge l \neq \text{ev} \Rightarrow l \in S] \end{aligned} \right) \\ & * \forall 1 \leq n' < n \leq M. [\text{seq}[n] \neq \text{seq}[n']] \\ & * [\text{hit} = 0] * [\text{ev} = u] \end{aligned} \right)$$

With propositional reasoning, we can replace $[\text{cache}[l][\text{value}] \neq h[l][\text{value}] \wedge l \neq \text{ev} \Rightarrow l \in S]$ with $[\text{cache}[l][\text{value}] \neq h[l][\text{value}] \Rightarrow l \in \{S \cup u\}]$ under the condition $[\text{ev} = u]$; also, because $\forall i < n \leq M. \forall 1 \leq l \leq k. [\text{seq}[n] \neq h[l][\text{value}]]$, we can strengthen the assertion $[l \in S \wedge l \neq \text{ev} \Rightarrow \text{cache}[l][\text{value}] \neq h[l][\text{value}]]$ into $[l \in S \cup \{u\} \Rightarrow \text{cache}[l][\text{value}] \neq h[l][\text{value}]]$. Thus, we have (102).

$$\begin{aligned}
& \left\{ \text{cost} = c * \left(\begin{array}{l} \forall 1 \leq n \leq M. \forall 1 \leq l \leq k. [\text{seq}[n] \neq h[l][\text{value}]] \\ * \forall 1 \leq l \leq k. [\text{cache}[l][\text{mark}] = 0] \\ \forall 1 \leq n' < n \leq M. [\text{seq}[n] \neq \text{seq}[n']] \end{array} \right) \right\} \\
& i := 1 \\
& \text{repeat } M: \\
& \quad \{P(i)\} \quad (97) \\
& \quad \text{hit} := 0 \\
& \quad j := 1 \\
& \quad \left\{ C_{K(i-1)} S. \left(\begin{array}{l} [\text{cost} = c + i - 1] * \forall i \leq n \leq M. \forall 1 \leq l \leq k. [\text{seq}[n] \neq h[l][\text{value}]] \\ * \forall l. [l \in S \Leftrightarrow \text{cache}[l][\text{mark}] = 1 \Leftrightarrow \text{cache}[l][\text{value}] \neq h[l][\text{value}]] \\ * [\text{hit} = 0] * [j = 1] * \forall 1 \leq n' < n \leq M. [\text{seq}[n] \neq \text{seq}[n']] \end{array} \right) \right\} \\
& \quad \text{repeat } k: \\
& \quad \quad \{Q(j)\} \quad (98) \\
& \quad \quad \text{if } \text{cache}[j][\text{value}] = \text{seq}[i]: \\
& \quad \quad \quad \text{cache}[j][\text{mark}] := 1; \text{hit} := 1 \\
& \quad \quad j := j + 1 \\
& \quad \left\{ C_{K(i-1)} S. \left(\begin{array}{l} [\text{cost} = c + i - 1] * \forall i \leq n \leq M. \forall 1 \leq l \leq k. [\text{seq}[n] \neq h[l][\text{value}]] \\ * \forall l. [l \in S \Leftrightarrow \text{cache}[l][\text{mark}] = 1 \Leftrightarrow \text{cache}[l][\text{value}] \neq h[l][\text{value}]] \\ * [\text{hit} = 0] * \forall 1 \leq n' < n \leq M. [\text{seq}[n] \neq \text{seq}[n']] \end{array} \right) \right\} \quad (99) \\
& \quad \text{if } \neg \text{hit} \text{ then:} \\
& \quad \quad \text{ev} \approx \text{Reservoir-Samp}(k, \text{cache}) \\
& \quad \quad \left\{ C_{K(i-1)} S. \left(\begin{array}{l} [\text{cost} = c + i - 1] * \forall i \leq n \leq M. \forall 1 \leq l \leq k. [\text{seq}[n] \neq h[l][\text{value}]] \\ * \forall l. [l \in S \Leftrightarrow \text{cache}[l][\text{mark}] = 1 \Leftrightarrow \text{cache}[l][\text{value}] \neq h[l][\text{value}]] \\ * \forall 1 \leq n' < n \leq M. [\text{seq}[n] \neq \text{seq}[n']] \\ * [\text{hit} = 0] * \text{ev} \sim \text{Unif}_{[1,k] \setminus S} \end{array} \right) \right\} \quad (100) \\
& \quad \quad \left\{ C_{K(i-1)} S. C_{\text{Unif}_{[1,k] \setminus S}} u. \left(\begin{array}{l} [\text{cost} = c + i - 1] * \forall i \leq n \leq M. \forall 1 \leq l \leq k. [\text{seq}[n] \neq h[l][\text{value}]] \\ * \forall l. [l \in S \Leftrightarrow \text{cache}[l][\text{mark}] = 1 \Leftrightarrow \text{cache}[l][\text{value}] \neq h[l][\text{value}]] \\ * \forall 1 \leq n' < n \leq M. [\text{seq}[n] \neq \text{seq}[n']] \\ * [\text{hit} = 0] * [\text{ev} = u] \end{array} \right) \right\} \quad (101) \\
& \quad \quad \text{cache}[\text{ev}][\text{value}] := \text{seq}[i] \\
& \quad \quad \left\{ C_{K(i-1)} S. C_{\text{Unif}_{[1,k] \setminus S}} u. \left(\begin{array}{l} [\text{cost} = c + i - 1] * (\forall i < n \leq M. \forall 1 \leq l \leq k. [\text{seq}[n] \neq h[l][\text{value}]]) \\ * (\forall l. [l \in S \Leftrightarrow \text{cache}[l][\text{mark}] = 1] \\ * [l \in S \cup \{u\} \Leftrightarrow \text{cache}[l][\text{value}] \neq h[l][\text{value}]] \\ * \forall 1 \leq n' < n \leq M. [\text{seq}[n] \neq \text{seq}[n']] \\ * [\text{hit} = 0] * [\text{ev} = u] \end{array} \right) \right\} \quad (102) \\
& \quad \quad \text{cache}[\text{ev}][\text{mark}] := 1 \\
& \quad \quad \left\{ C_{K(i-1)} S. C_{\text{Unif}_{[1,k] \setminus S}} u. \left(\begin{array}{l} [\text{cost} = c + i - 1] * \forall i \leq n \leq M. \forall 1 \leq l \leq k. [\text{seq}[n] \neq h[l][\text{value}]] \\ * \forall l. [l \in (S \cup \{u\}) \Leftrightarrow \text{cache}[l][\text{mark}] = 1 \Leftrightarrow \text{cache}[l][\text{value}] \neq h[l][\text{value}]] \\ * \forall 1 \leq n' < n \leq M. [\text{seq}[n] \neq \text{seq}[n']] \\ * [\text{hit} = 0] * [\text{ev} = u] \end{array} \right) \right\} \\
& \quad \quad \left\{ C_{K(i)} S. \left(\begin{array}{l} [\text{cost} = c + i - 1] * \forall i + 1 \leq n \leq M. \forall 1 \leq l \leq k. [\text{seq}[n] \neq h[l][\text{value}]] \\ * (\forall l. [l \in (S \cup \{u\}) \Leftrightarrow \text{cache}[l][\text{mark}] = 1 \Leftrightarrow \text{cache}[l][\text{value}] \neq h[l][\text{value}]] \\ * \forall 1 \leq n' < n \leq M. [\text{seq}[n] \neq \text{seq}[n']] \end{array} \right) \right\} \quad (104) \\
& \quad \quad \text{cost} := \text{cost} + 1 \\
& \quad \quad \left\{ C_{K(i)} S. \left(\begin{array}{l} [\text{cost} = c + i] * \forall i + 1 \leq n \leq M. \forall 1 \leq l \leq k. [\text{seq}[n] \neq h[l][\text{value}]] \\ * (\forall l. [l \in (S \cup \{u\}) \Leftrightarrow \text{cache}[l][\text{mark}] = 1 \Leftrightarrow \text{cache}[l][\text{value}] \neq h[l][\text{value}]] \\ * \forall 1 \leq n' < n \leq M. [\text{seq}[n] \neq \text{seq}[n']] \end{array} \right) \right\} \\
& \quad i := i + 1 \\
& \quad \left\{ C_{K(M)} S. \left(\begin{array}{l} [\text{cost} = c + M] * \forall 1 \leq n' < n \leq M. [\text{seq}[n] \neq \text{seq}[n']] \\ * \forall l. [l \in S \Leftrightarrow h[l][\text{mark}] = 1 \\ \Leftrightarrow \text{cache}[l][\text{value}] \neq h[l][\text{value}]] \end{array} \right) \right\}
\end{aligned}$$

Fig. 18. Cache: clean phase

Similarly, applying **WP-ASSIGN** and using propositional reasoning, we can derive (103). For the step (104), we apply **C-FUSE** to combine $C_{\kappa(i-1)} S. C_{\text{Unif}_{[1,k] \setminus S}} u.$ into $C_{\kappa(i-1) \prec \text{Unif}_{[1,k] \setminus S}}(S, u)$. We then apply **C-TRANSF** based on the bijection between (S, u) and $(S \cup \{u\}, u)$ to get that such that

$$C_d(S', u) \cdot \left(\begin{array}{l} \text{cost} = c + i - 1 * \forall i + 1 \leq n \leq M. \forall 1 \leq l \leq k. [\text{seq}[n] \neq h[l][\text{value}]] \\ * (\forall l. [l \in S' \Leftrightarrow \text{cache}[l][\text{mark}] = 1 \Leftrightarrow \text{cache}[l][\text{value}] \neq h[l][\text{value}]] \\ * \forall 1 \leq n' < n \leq M. [\text{seq}[n] \neq \text{seq}[n']] \end{array} \right),$$

where

$$\begin{aligned} d &= \text{bind}(\kappa(i-1), \lambda S. \text{bind}(\text{Unif}_{[1,k] \setminus S}, \lambda u. \delta_{(S \cup \{u\}, u)))) \\ &= \text{bind}(\kappa(i), \lambda S. \text{bind}(\text{Unif}_{[1,k] \setminus S}, \lambda u. \delta_{(S, u)})) \end{aligned}$$

Since u is not used, we could use **C-SURE-PROJ** to project out u and derive that

$$C_{\kappa(i)} S' \cdot \left(\begin{array}{l} \text{cost} = c + i - 1 * \forall i + 1 \leq n \leq M. \forall 1 \leq l \leq k. [\text{seq}[n] \neq h[l][\text{value}]] \\ * (\forall l. [l \in S' \Leftrightarrow \text{cache}[l][\text{mark}] = 1 \Leftrightarrow \text{cache}[l][\text{value}] \neq h[l][\text{value}]] \\ * \forall 1 \leq n' < n \leq M. [\text{seq}[n] \neq \text{seq}[n']] \end{array} \right)$$

Then, by **WP-ASSIGN** and **WP-LOOP**, we can derive the post-condition

$$C_{\kappa(M)} S \cdot \left(\begin{array}{l} \text{cost} = c + M * \forall 1 \leq n' < n \leq M. [\text{seq}[n] \neq \text{seq}[n']] \\ * \forall l. [l \in S \Leftrightarrow \text{cache}[l][\text{mark}] = 1 \Leftrightarrow \text{cache}[l][\text{value}] \neq h[l][\text{value}]] \end{array} \right)$$

Last, with **SURE-STR-CONVEX**, we can establish

$$[\text{cost} = c + M] * C_{\kappa(M)} S. \forall l. [l \in S \Leftrightarrow \text{cache}[l][\text{mark}] = 1 \Leftrightarrow \text{cache}[l][\text{value}] \neq h[l][\text{value}]]$$

which states our goal that cost increases by exactly M and the set of evicted indices S is picked uniformly.