Roulette: A Language for Expressive, Exact, and Efficient Discrete Probabilistic Programming (with Appendices)

CAMERON MOY, Northeastern University, USA
JACK CZENSZAK, Northeastern University, USA
JOHN M. LI, Northeastern University, USA
BRIANNA MARSHALL, Northeastern University, USA
STEVEN HOLTZEN, Northeastern University, USA

Exact probabilistic inference is a requirement for many applications of probabilistic programming languages (PPLs) such as in high-consequence settings or verification. However, designing and implementing a PPL with scalable high-performance exact inference is difficult: exact inference engines, much like SAT solvers, are intricate low-level programs that are hard to implement. Due to this implementation challenge, PPLs that support scalable exact inference are restrictive and lack many features of general-purpose languages.

This paper presents Roulette, the first discrete probabilistic programming language that combines high-performance exact inference with general-purpose language features. Roulette supports a significant subset of Racket, including data structures, first-class functions, surely-terminating recursion, mutable state, modules, and macros, along with probabilistic features such as finitely supported discrete random variables, conditioning, and top-level inference. The key insight is that there is a close connection between exact probabilistic inference and the symbolic evaluation strategy of Rosette. Building on this connection, Roulette generalizes and extends the Rosette solver-aided programming system to reason about probabilistic rather than symbolic quantities. We prove Roulette sound by generalizing a proof of correctness for Rosette to handle probabilities, and demonstrate its scalability and expressivity on a number of examples.

CCS Concepts: • **Mathematics of computing → Probabilistic inference problems**.

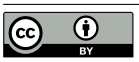
Additional Key Words and Phrases: probabilistic programming, symbolic evaluation

ACM Reference Format:
Cameron Moy, Jack Czenszak, John M. Li, Brianna Marshall, and Steven Holtzen. 2025. Roulette: A Language for Expressive, Exact, and Efficient Discrete Probabilistic Programming (with Appendices). *Proc. ACM Program. Lang.* 9, PLDI, Article 231 (June 2025), 81 pages. https://doi.org/10.1145/3729334

1 Efficient Inference With Improved Expressivity

There are many applications of probabilistic programming languages (PPLs) that require or benefit from deterministic exact answers to probabilistic inference queries. In some kinds of verification, such as verified differential privacy [2, 41], it is critical that probabilistic inference be exact because even a small probability of error can be catastrophic. Even when approximations are permissible, exact inference is often useful as a subroutine for scaling approximate inference using Rao-Blackwellization [37].

Authors' Contact Information: Cameron Moy, Northeastern University, Boston, USA, camoy@ccs.neu.edu; Jack Czenszak, Northeastern University, Boston, USA, czenszak.j@northeastern.edu; John M. Li, Northeastern University, Boston, USA, li.john@northeastern.edu; Brianna Marshall, Northeastern University, Boston, USA, marshall.br@northeastern.edu; Steven Holtzen, Northeastern University, Boston, USA, s.holtzen@northeastern.edu.



ACM 2475-1421/2025/6-ART231
https://doi.org/10.1145/3729334

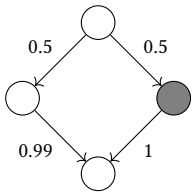
Unfortunately, implementing PPLs that support scalable high-performance exact inference is difficult today and requires great care and expertise. Inference is the task of computing the probability that a probabilistic program evaluates to a particular value, and is #P-hard even for restricted languages (e.g., straight-line programs with only `if`-statements and Boolean random variables [29, 53]). Consequently, state-of-the-art exact inference algorithms walk the fine line of computational intractability. Much like SAT solvers, they rely on careful heuristics and performance tricks to maximize performance at the edge of worst-case hardness. This makes them delicate and error-prone programs—not something one wants to implement twice.

The consequence of this difficulty is that today's languages that support exact probabilistic inference are either (1) small purpose-built languages that support only a limited subset of language features, or (2) expressive but lag behind the performance of state-of-the-art approaches that target more restricted languages. In the first category are languages such as Dice [5, 21, 29], ProbLog [17], and SPPL [55]: these languages are quite restrictive and omit most high-level features (e.g., recursion and mutable state), but in exchange can use high-performance exact inference algorithms. In the second category are languages such as PSI [23, 24] and Hakaru [38]: these languages are more expressive and support many language features (e.g., continuous random variables), but their performance lags behind the more restricted languages when they are evaluated on head-to-head benchmark problems [23, 29].

Our aim is to narrow the exact inference performance gap between expressive PPLs and restricted PPLs. Towards this goal, we present Roulette, the first discrete PPL that supports high-performance exact probabilistic inference and the essential features of a general-purpose language. Specifically, Roulette is a PPL with finitely supported discrete random variables, Bayesian conditioning, and a significant subset of the Racket programming language [18], including its data structures, standard library, and macro system. Roulette brings together two main ideas: *exact inference via knowledge compilation* [7, 14, 17, 29, 56] and *execution via symbolic evaluation* [60, 61].

Knowledge compilation (KC) is the state-of-the-art approach for exact inference of probabilistic graphical models and discrete probabilistic programs [7, 9, 29, 43, 56]. The heart of KC is a reduction from probabilistic inference to *weighted model counting* (WMC), which is a probabilistic analogue of the satisfiability problem: given a Boolean formula φ and weight map w that maps literals to real-valued weights, $\texttt{WMC}(\varphi, w)$ computes the cumulative weight of models of φ. Steady progress has been made on designing and implementing scalable WMC solvers, and today these tools are widely used within the AI and automated reasoning communities for solving discrete probabilistic inference tasks [32, 35, 44, 45].

The key challenge with applying inference-via-KC to probabilistic program inference is the laborious translation from programs to weighted Boolean formulae. Existing PPLs that leverage inference-via-KC, such as Dice [29], meticulously translate each language feature into a weighted Boolean formula and prove each translation correct. Due to the degree of manual work involved in this translation, Dice is an impoverished language. An analogous challenge of encoding programs into logical formulae is present in traditional symbolic execution, and herein lies a solution: the Rosette solver-aided programming model [60, 61]. Rosette is a programming language and symbolic evaluator within the Racket ecosystem. Rosette allows developers to write expressive high-level Racket code enhanced with primitives for introducing and constraining symbolic values. It then compiles programs into SMT constraints and dispatches them to an off-the-shelf SMT solver such as Z3 [15]. The results of queries are then made available from within the language. Rosette has proven effective in practice and has found numerous applications [10, 39, 64]: programmers can embed a language in Rosette and immediately inherit a state-of-the-art symbolic evaluator, without manually encoding each language construct as an SMT constraint.



(a) The Diamond Network Visualized

```
(define SIZE 4)

(define (diamond data)
  (if (flip 0.5) (noisy 0.01 data) (drop 0.99 data)))

(define (drop rate node)
  (if (flip rate) node #false))

(define (noisy rate data)
  (define bit (select (range SIZE)))
  (define mask (rotate-left bit (bv 1 SIZE)))
  (if (and (flip rate) data) (bvxor data mask) data))
```

(b) Definition of `diamond`

Fig. 1. Network Reliability Example

Roulette generalizes Rosette to handle finitely supported discrete random variables rather than symbolic variables, enabling high-performance inference-via-KC for a significant subset of Racket. The rest of this paper proceeds as follows. First, Section 2 gives a quick tour of Roulette and demonstrates its unique expressivity by computing network analyses, performing inference on Bayesian networks, running an interpreter for unreliable hardware, exploring laziness for geometric distributions, and embedding a relational language. Then, Section 3 gives an informal semantics of Roulette, explains how Roulette compiles programs to weighted Boolean formulae, and articulates key points that differentiate Roulette from Rosette. Sections 4 and 5 give a formal semantics for Rosette and Roulette, respectively, and prove correctness theorems for both. Section 6 empirically evaluates the performance of Roulette on a number of benchmarks, including well-known Bayesian networks and tasks that are traditionally challenging for exact inference. The benchmarks establish that, when suitably optimized, Roulette is capable of competitive performance with existing state-of-the-art exact inference strategies while being significantly more expressive. Section 7 discusses related work and Section 8 concludes.

2 Programming in Roulette

This section presents examples that demonstrate some unique features of Roulette, a subset of Racket augmented with discrete random variables, observation, and top-level inference. All the usual constructs of a traditional Scheme-like functional language, including symbols, lists, recursive functions, mutable state, standard library functions, macros, and REPL interactions, are available. Each example is intended to showcase some of these features.

2.1 Data Structures for Network Reliability

Many important problems require exact discrete probabilistic inference. One example is network reliability, which asks for the probability that a packet reaches some node given a network topology and some chance of failure along the way. PPLs are useful for modeling such tasks because one can use a probabilistic program to describe the network's topology and randomized routing protocols [22, 29, 57]. Then, once the network is described, the PPL can answer queries about the network's behavior, such as the probability that a packet successfully traverses the network with its data intact. In this setting, exact inference is critical because failure probabilities can be small and errors catastrophic [57].

Figure 1 gives an example encoding of a network reliability problem in Roulette. Figure 1a shows a visualization of a small network where nodes in the graph represent routers and edges represent

directed links between routers. Edges are annotated with the probability a packet is forwarded along that connection, and the gray router is faulty and may corrupt the payload. Figure 1b shows a program that models this network's behavior, written in Roulette. The `diamond` function, given a bitvector payload representing a packet, returns the payload when it reaches the end of the network or `#false` if the packet is dropped. The probabilistic behavior in this program is introduced via the `(flip p)` syntax, which yields a Boolean random variable that is `#true` with probability p and `#false` with probability $1 - p$. The `diamond` function forwards the packet to one of two routers with 50% probability: the behavior of these two routers is modeled by the `drop` and `noisy` functions. The `drop` function drops a packet with some probability. The `noisy` function forwards a packet and potentially introduces some error in the payload by drawing a bit index uniformly between `0` and `SIZE-1` (using `select`), masking with 1 at that index, and then computing the exclusive-or of the given payload with probability `rate`. In other words, `noisy` randomly flips a bit of the payload with probability `rate`.

Here is an example output from the program, printed using Roulette's REPL:

```
> (diamond (bv #b1111 SIZE))
(pmf | #false        : 0.00500 | (bv #b1111 4) : 0.99
     | (bv #b0111 4) : 0.00125 | (bv #b1011 4) : 0.00125
     | (bv #b1101 4) : 0.00125 | (bv #b1110 4) : 0.00125)
```

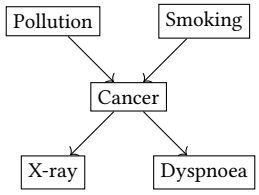
The above probability mass function (`pmf`) maps Racket values to their probabilities. The most likely returned value for the network program is `(bv #b1111 4)`, which denotes a successful traversal. Note that the program is written without contortions, using ordinary language features that would be familiar to Racket or Scheme programmers: conditionals, functions, and lists. Crucially, despite being written in a high-level style, Roulette efficiently and exactly calculates the above PMF—even for large networks. See Section 6 for details.

2.2 Recursion and Interoperability for Bayesian Networks

A Bayesian network is a probabilistic graphical model that represents a set of variables and the causal relationships between them [46]. Bayesian networks are widely used probabilistic models with well-studied inference algorithms, so they make a good baseline for measuring the performance of PPLs. Bayesian network nodes are random variables and directed edges indicate dependencies. Every node is associated with a table of probabilities enumerating the probability of each possible value, conditioned on all dependent variables. Querying a Bayesian network may, for example, amount to computing the marginal probability of some variable.

Consider the Cancer network shown graphically in Figure 2a. There are four variables shown as nodes: Pollution, Smoking, Cancer, X-ray, and Dyspnoea. Each variable has a directed edge to variables that it may influence. So, lung cancer is directly influenced by whether someone smokes and is exposed to pollution. In turn, an individual's X-ray result and the presence of dyspnoea (shortness of breath) are affected by lung cancer. Each variable is associated with a table that lists the probability of every possible value conditioned on some assignment to dependent variables. For example, there are three levels of Pollution: Low (0), Medium (1), and High (2). The table for the Cancer variable is also shown in Figure 2a, where the first row states that the probability of lung cancer is 0.03 conditional on low pollution and smoking.

Roulette can interpret Bayesian networks by generating categorical random variables for each node, conditioned on all causal dependencies. The code for doing so is provided in Figure 2b. Given an s-expression representation of a Bayesian network, the `make-network` function produces a hash table that maps variables to nodes. Each node contains a random variable associated with that



Pollution	Smoking	Cancer (True)
Low	True	0.03
Low	False	0.001
Medium	True	0.03
Medium	False	0.001
High	True	0.05
High	False	0.02

(a) Definition

```
(struct node (value size))

(define (make-network data)
  (for/fold ([acc (hash)])
            ([elem (in-list data)])
    (match-define (list name deps rows ...) elem)
    (hash-set acc name (make-node acc deps rows))))

(define (make-node acc deps rows)
  (match deps
    [(list)
     (match-define (list row) rows)
     (node (categorical row) (length row))]
    [(cons dep deps*)
     (match-define (node var size) (hash-ref acc dep))
     (define rows* (list-ref (slice-at rows size) var))
     (make-node acc deps* rows*)]))
```

(b) Builder

Fig. 2. Bayesian Network Example

node. Exactly how `make-network` works is not so important, just note that the code is short and idiomatic—using features such as recursion, hash tables, pattern matching, and for comprehensions.

The `expectation` function can be used to compute the expected value of a random variable from the Bayesian network:

```
> (define net (make-network (read-bif)))
> (expectation (node-value (hash-ref net 'Pollution)))
0.6
```

First, `net` is defined by parsing a Bayesian network in the Bayesian Interchange Format (BIF) from `STDIN`. The BIF parser is written in Racket and uses its lexing and parsing facilities—Roulette can seamlessly interoperate with Racket so long as concrete values, and not probabilistic quantities, cross the language boundary. For instance, an ordinary Racket function should not be applied to `(flip 0.5)`, but certainly can be applied to the constant value `#true`. Next, the expectation of a variable is computed: given a random variable that always takes on a numeric value, `expectation` computes its weighted average. Expectation is an example of *top-level inference*, where a program can perform inference at any point during execution and use the result in subsequent computations.

Expectation can be combined with other probabilistic operations such as conditioning:

```
> (observe! (equal? (node-value (hash-ref net 'Cancer)) 1))
> (expectation (node-value (hash-ref net 'Pollution)))
0.83233...
```

Observation lies at the heart of Bayesian reasoning. In the standard Bayesian workflow, a programmer: (1) defines a prior probability distribution, (2) conditions on some evidence, and (3) computes a posterior probability distribution. To accommodate this style of reasoning, PPLs often provide an `observe!` construct that conditions on evidence. For a discrete language, `observe!` takes a value and eliminates all worlds in which that value is `#false`. So, the above code calculates the expected level of pollution given that an individual has lung cancer. Intuitively, if a patient is diagnosed with lung cancer, then the expected amount of pollution that patient has been exposed to should increase. This intuition is borne out by the data since 0.83233 is greater than 0.6.

```
(define BIT-WIDTH 16)
(define MEMORY-SIZE 2)

(define (blank-memory)
  (list->vector
    (build-list MEMORY-SIZE
      (λ _ (bv-zero BIT-WIDTH)))))

(define (load mem a)
  (define i (bv->number a))
  (mem-perturb mem i)
  (vector-ref mem i))

(define (store mem a v)
  (define i (bv->number a))
  (mem-perturb mem i)
  (vector-set! mem i v)
  mem)

(define (mem-perturb mem i)
  (define v
    (if (flip 0.0001)
        (bv 0 BIT-WIDTH)
        (vector-ref mem i)))
  (vector-set! mem i v))
```

(a) Memory Model

```
(define ((read e) m) (load m (e m)))
(define ((int i) _) (bv i BIT-WIDTH))
(define ((add e1 e2) m) (bvadd (e1 m) (e2 m)))
(define (skip k m) m)
(define ((seq c1 c2) k m) (c2 k (c1 k m)))
(define ((asgn e1 e2) k m) (store m (e1 m) (e2 m)))
(define ((ite e c1 c2) k m)
  (if (bv-nonzero? (e m)) (c1 k m) (c2 k m)))
(define ((while e c) k m)
  (cond
    [(= k 0) #false]
    [(bv-nonzero? (e m)) ((seq c (while e c)) (- k 1) m)]
    [else m]))
```

(b) Definitional Interpreter

```
(define (tri n)
  (define total (int 0))
  (define i (int 1))
  (seq (asgn i (int n))
       (while (read i)
         (seq (asgn total (add (read total) (read i)))
              (asgn i (add (read i) (int -1)))))))

(define (read-total m)
  (if m (vector-ref m 0) #false))
```

(c) Test Program

Fig. 3. Unreliable Hardware Example

2.3 First-Class Functions and Mutation for Unreliable Hardware

This subsection demonstrates how higher-order functions and mutable state can be used in tandem to analyze the behavior of imperative programs under a low-level execution model where loads and stores from memory are unreliable. The implementation is shown in Figures 3a and 3b. Figure 3a defines the execution model as a machine with 16-bit words and two cells of addressable memory.[1] The memory is represented as a mutable vector of length `MEMORY-SIZE`, and each item in the vector is a bitvector of length `BIT-WIDTH`. The functions `load` and `store` model loads and stores from an unreliable memory. These functions are defined in terms of `vector-ref` and `vector-set!`, which read from and write to entries of a vector, and the `mem-perturb` function. Using `mem-perturb` introduces unreliability: every load and store from memory cell i has a 0.0001 chance of corrupting it, zeroing out its contents.

Figure 3b builds on this model of unreliable memory with a definitional interpreter for a simple imperative language with arithmetic, assignment statements, conditional branching, and while loops. Since Roulette supports higher-order functions and recursion, the interpreter is completely standard, following the usual semantics of WHILE programs. Arithmetic expressions are modelled as functions that take in a memory and produce a bitvector: `(read e)` loads the contents of the address denoted by `e` from memory, `(int i)` produces the two's-complement representation of the integer `i`, and `(add e1 e2)` adds the results produced by `e1` and `e2`. Commands are modelled by functions that receive a fuel parameter `k` and a memory `m`, and produce either an updated memory or the constant `#false` signalling that the interpreter ran out of fuel. Augmenting an interpreter with fuel is a standard technique for ensuring termination [42]. The functions `skip` and

[1] Two cells suffice for the example (Figure 3c); it is easy to add more.

`seq` interpret sequencing of commands, `asgn` interprets assignment, `ite` interprets branching, and `while` interprets while loops.

Finally, Figure 3c uses the interpreter from Figure 3b to define a simple WHILE program `(tri n)` that computes the sum $1 + \cdots + n$ and stores the answer at address 0. Due to the unreliability of memory, there is a chance `(tri n)` produces the wrong result. Nonetheless, since the probability of hardware failure is small, the correct result should be computed with high probability:

```
> (bv->number (read-total ((tri 10) 12 (blank-memory))))
(pmf | 55 : 0.996107 | 45 : 0.000199 | ... | #false : 0.001996)
```

Shown in the output is the distribution over possible values stored at address 0 after running the program `(tri 10)` with twelve units of fuel. The (abbreviated) probability mass function for this distribution looks as expected: the correct result (55) has the highest probability, some erroneous results (e.g., 45) caused by hardware failure have small probabilities, and some execution traces do not terminate within the amount of fuel provided. This example is challenging due to the high-dimensional state space. Section 6 shows that Roulette significantly outperforms naive exhaustive enumeration on these programs.

2.4 Macros and Laziness for Geometric Distributions

The geometric distribution models the number of failed Bernoulli trials before the first success. An example of a geometric distribution involves flipping a coin repeatedly until it lands heads up for the first time, then counting how many times the coin previously landed tails up. This experiment can be expressed as a recursive program:

```
(define (geom p)
  (if (flip p) 0 (+ 1 (geom p))))
```

Unfortunately, any program that applies `geom` will not surely terminate. As mentioned previously, Roulette works only with finitely supported discrete random variables and programs that surely terminate. At first glance, it seems geometric distributions are out of reach.

However, programmers in conventional languages can employ *laziness* to transform some non-terminating programs into terminating ones. That insight is applicable here too, by leveraging the expressiveness of Roulette. If the recursive call is placed inside a thunk, only as many levels of recursion as needed to answer *finite* queries about the distribution have to be computed.

This insight implies that the expression `(+ 1 (geom p))` cannot be eagerly evaluated. Instead, lazy Peano numbers [54] can encode partially evaluated natural numbers.[2] A program can redefine the `+` operator as a macro that thunks the second argument by using the `delay` macro from Racket's promise library and then use Peano addition:

```
(define-syntax-rule (+ n m)
  (peano-add n (delay m)))
```

The `geom` function, without modifying its source code, can now terminate under this new definition of `+` (assuming a suitable implementation of Peano numerals). Well-known properties of the geometric distribution, such as its probability mass and cumulative distribution functions, can be computed using `=` and `<=`, respectively:

```
> (= (geom 0.5) 3)
(pmf | #true : 0.0625 | #false : 0.9375)
> (<= (geom 0.2) 5)
(pmf | #true : 0.7379 | #false : 0.2621)
```

[2]Peano numerals are not an efficient representation, but there are suitable alternatives [28] that may perform better.

```
(define e1 (flip 0.6))
(define e2 (flip 0.1))
(define e3 (flip 0.4))
(define e4 (flip 0.3))

(define-relation (edgeo x y)
  (disj
    (if e1 (== (cons x y) (cons 'A 'B)) fail)
    (if e2 (== (cons x y) (cons 'A 'C)) fail)
    (if e3 (== (cons x y) (cons 'B 'D)) fail)
    (if e4 (== (cons x y) (cons 'C 'D)) fail)))
```

(a) An Example Network

```
> (define-relation (patho x z)
    (disj
      (edgeo x z)
      (fresh (y)
        (conj (edgeo x y)
              (patho y z)))))

> (run* x (patho 'A 'D))
(pmf | (list)           : 0.7372
     | (list '_.0)      : 0.2556
     | (list '_.0 '_.0) : 0.0072)
```

(b) Definition and Example of `patho`

Fig. 4. μKanren Example

The sum of geometric distributions has a closed form, but the product of geometric distributions does not. Exact inference is often more challenging in situations where there is no closed form solution. Since Roulette's inference strategy does not rely on the existence of closed form solutions, it can handle products just fine:

```
> (<= (* (geom 0.5) (geom 0.2)) 8)
(pmf | #true : 0.8572 | #false : 0.1428)
```

In short, non-probabilistic expressiveness can increase probabilistic expressiveness by allowing programmers to describe non-finitely supported random variables via laziness.

2.5 Embedded Relational Programming

As shown in Section 2.3, Roulette is expressive enough to embed interpreters for other languages. Going one step further, Roulette can be turned into a *probabilistic logic programming language* [16, 20, 40] by embedding a traditional logic programming language implementation. Probabilistic logic programming mixes traditional logic programming and probabilistic programming by making rules or atoms uncertain. Querying in this setting involves not only determining whether a goal unifies but also the probability it does so. Traditionally, implementing probabilistic logic programming languages has involved custom solutions for translating programs into a logical formula for inference: for instance, a state-of-the-art implementation of ProbLog compiles programs into weighted Boolean formulae for inference [17]. In Roulette, it is easy to get a probabilistic logic programming language by *embedding* a non-probabilistic interpreter for a relational language.

A good candidate for such an exercise is μKanren [26, 27]. The Scheme community has for many years developed relational programming languages, the most well known being miniKanren [4, 19]. Relational programming differs from traditional logic programming in a few ways. For instance, the interleaving search strategy of miniKanren differs from Prolog's depth-first search. Another difference, and the one most relevant here, is that miniKanren and its various flavors are often embedded—frequently in Scheme. In particular, one small relational language called μKanren falls neatly within the features Roulette supports. Therefore, the implementation of μKanren can quite literally be copy-and-pasted into a Roulette program and run *unmodified*.

Consider a network reliability problem, similar to the one from Section 2.1 but without noisy packets. Determining whether a packet successfully traverses a network requires encoding the network as a relation on edges. Figure 4a defines a relation `edgeo` that succeeds if the given nodes have an edge between them. Given two inputs, `edgeo` unifies if any of the goals inside the disjunction operation `disj` unify. Each goal inside `disj` corresponds to an edge and unifies, via ==, with the given edge probability.

```
(define (search xs v)
  (cond
    [(empty? xs) (list)]
    [(equal? (first xs) v) xs]
    [else (search (rest xs) v)]))
```

(a) Definition

```
> (search 3 (list 2 3 4 3))
(list 3 4 3)

> (search 1 (list 2 3 4 3))
(list)
```

(b) Example

Fig. 5. The `search` Function

Here is an example μKanren query evaluated using Roulette:

```
> (run* x (edgeo 'A x))
(pmf | (list) : 0.36    | (list 'B) : 0.54
     | (list 'C) : 0.04 | (list 'B 'C) : 0.06)
```

The `run*` form returns all possible substitutions for `x` that unify `(edgeo 'A x)`. Running in Roulette yields a distribution over substitutions. Note how this example freely mixes features of Roulette with μKanren: first, it uses Roulette's built-in `flip` construct to introduce random variables. Then, it uses μKanren's built-in `define-relation` construct to relate these random variables and `run*` to query them as normal. From the perspective of the embedded μKanren implementation, it is as if the edge variables are regular Racket Booleans.

Continuing with the example in Figure 4b, reachability is defined to be the transitive closure of `edgeo`, and then `run*` is used to determine the probability that two nodes are reachable. Here, `(patho x z)` unifies if either there is an edge between `x` and `z` or there is some intermediate node `y` connecting `x` to `z`. Defining this second goal involves two μKanren forms: `fresh`, which generates a new logic variable, and `conj`, which unifies if all its goals do too. The results contain `'_.0` because the logic variable `x` remains unconstrained after the goal unifies. There are three substitutions in the final distribution: the empty list (unreachable), the singleton list (only one path from A to D), and the two-element list (two paths from A to D). So, node A reaches node D with probability $0.2556 + 0.0072 = 0.2628$.

This example is *not* intended to demonstrate that embedding μKanren yields a state-of-the-art probabilistic logic programming language. The μKanren implementation is fewer than 40 lines of code and is intended for pedagogic purposes. There should be no expectation that this implementation competes with a specialized compiler. However, it does showcase Roulette's expressivity and suggests further experimentation with high-performance miniKanren compilers [1].

3 How Roulette Performs Scalable Inference, by Example

Roulette derives its expressivity from Rosette's symbolic semantics and its performance from a weighted model counting backend based on binary decision diagrams (BDDs), similar to Dice [29]. This section informally summarizes (1) how Rosette symbolically evaluates programs into formulae, and then (2) how Roulette adapts Rosette to generate weighted Boolean formulae for inference.

3.1 How Rosette Efficiently Symbolically Evaluates Racket Programs

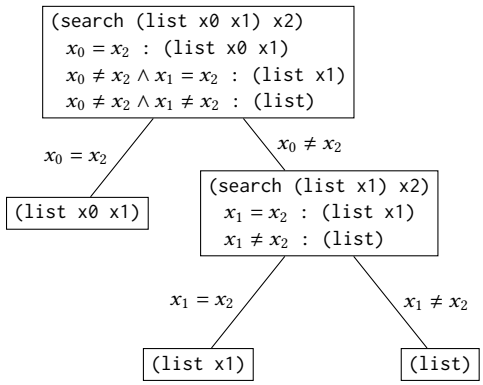
Consider the Racket definition of `search` in Figure 5a. Given a list and an element to look for, `search` returns either a suffix of the input list starting with the desired element or an empty list if the element is not present. An example REPL interaction is given in Figure 5b. Rosette can symbolically evaluate this program to check that it satisfies various correctness properties: for instance, Rosette can verify that for an input list of length n, `search` always returns another list of length at most n.

```
> (define-symbolic x0 integer?)
> (define-symbolic x1 integer?)
> (define-symbolic x2 integer?)
> (define result (search (list x0 x1) x2))
(union
  [x0 = x2 (list x0 x1)]
  [x0 ≠ x2 ∧ x1 = x2 (list x1)]
  [x0 ≠ x2 ∧ x1 ≠ x2 (list)])

> (verify (assert (<= (length result) 2)))
(unsat)

> (verify (assert (< (length result) 2)))
(model [x0 0] [x1 1] [x2 0])
```

(a) REPL Example



(b) Visualizing the Symbolic Union

Fig. 6. Verification of `search` in Rosette

Figure 6a shows how Rosette verifies this fact for lists of length 2. First, three fresh symbolic integers are created using the Rosette form `define-symbolic`. Next, `search` is called using these symbolic integers as arguments. The value returned by `search`, stored in `result`, is a *symbolic union* that represents all possible behaviors of `search` on the given inputs. Symbolic unions are sequences of formula–value pairs representing possible values and the conditions under which each value is produced. The `result` symbolic union in this example is a list where each element captures one of the three possible symbolic return values of the `search` program. For example, for the first element, if the guard $x_0 = x_2$ is true, then `search` evaluates to `(list x0 x1)`.

One can now check that `search` does not return a list that is longer than its input list by calling `verify` with the assertion that the length of the result is less than or equal to 2. Rosette outputs `(unsat)`, meaning that no counterexample to this assertion was found. If <= is changed to < instead, then the program outputs a model describing instantiations of the symbolic variables that produce a counterexample.

Rosette is designed to efficiently generate symbolic unions without suffering from path explosion. This process is visualized in Figure 6b, which gives an abstract view of how the symbolic union placed in `result` is calculated. If the guard of a `cond` depends on a symbolic value, Rosette executes both branches. For example, in `(search (list x0 x1) x2)` the first such guard is `(equal? (first xs) v)` which can be either true or false depending on the instantiation of the symbolic values x_0 and x_2. Thus, both branches are executed, and the results are *merged*. Symbolic unions are merged by conjoining the guard formula to each component of the union. Merging avoids the path explosion that is common in traditional symbolic execution. By exploiting the logical structure present in the symbolic unions, Rosette can scale to programs with many paths. Often, the bottleneck is the SMT solver rather than Rosette itself [10, 60]. For this example, the formula $x_0 = x_2$ is conjoined in the success branch, and $x_0 \neq x_2$ is conjoined in the failure branch.

Rosette's symbolic evaluation strategy is able to directly reuse host language functions, such as `length`, by lifting functions to work over symbolic unions. Roughly speaking, lifting involves mapping the operation over all elements of the union. Only primitive operations on symbolic values, such as `equal?`, must be modified to encode formulae. As such, most language features from the host, i.e. Racket, are automatically available.

3.2 How Roulette Generalizes Rosette to Probabilities

Getting Rosette to perform probabilistic inference requires two additional steps: (1) convert formulae to a representation that is appropriate for an external inference engine based on weighted model counting and (2) query the engine for probabilities. Roulette extends Rosette with an `infer` function that takes a symbolic value and returns a PMF mapping concrete values to probabilities. Rosette's existing compilation pipeline can otherwise stay exactly the same. Hence, Roulette benefits from the significant engineering effort put into Rosette's formula simplification and related optimizations.

The following is an example Roulette program that looks quite similar to the Rosette example in Figure 6a, but makes use of probabilistic rather than symbolic quantities:

```
> (define-symbolic x0 (bern 0.5))
> (define-symbolic x1 (bern 0.5))
> (define result (+ (if x0 2 3) (if x1 3 4)))
(union [x0 ∧ x1 5] [(¬x0 ∧ x1) ∨ (x0 ∧ ¬x1) 6] [¬x0 ∧ ¬x1 7])

> (infer result)
(pmf | 5 : 0.25 | 6 : 0.50 | 7 : 0.25)
```

First, two symbolic Booleans are created and associated with a probability distribution. Here, `x0` and `x1` are fair and independent coin flips. Next, the integer `result` is formed depending on the outcome of these two coin flips. Notice that the `result` symbolic union is exactly the same as it would be in Rosette if `x0` and `x1` were ordinary symbolic Booleans. Finally, `infer` is given `result`, and the PMF is displayed in tabular format. The examples in Section 2 work with more high-level language constructs than `define-symbolic`; these are desugared away using Racket macros, described further in Appendix A. To evaluate `infer`, Roulette computes the probability that the guard of each symbolic union is true. For example, the probability that `result` evaluates to 5 is given by the probability that $x_0 \wedge x_1$ is true, where x_0 and x_1 are independent Bernoulli random variables that are each true with probability 0.5. The correct answer is 0.25, which is indeed the probability printed in the REPL that corresponds to this outcome.

Roulette performs weighted model counting to calculate probabilities. So, to compute the probability that `result` evaluates to 6 in the above example, one must compute the probability that the symbolic union guard $\varphi = (\neg x_0 \wedge x_1) \vee (x_0 \wedge \neg x_1)$ is true. Given a weight map $w(x_0) = w(\neg x_0) = w(x_1) = w(\neg x_1)$, the following holds:

$$\Pr(\varphi) = \mathtt{WMC}(\varphi, w) = w(\neg x_0)w(x_1) + w(x_0)w(\neg x_1) = 0.5,$$

which matches the probability in the above PMF. Efficiently computing this weighted model count relies on knowledge compilation: these Boolean formulae are compiled into binary decision diagrams, which admit a straightforward linear-time weighted model counting procedure using dynamic programming. Figure 7 shows the BDD that represents the formula φ. Each node is annotated with a Boolean variable. Solid edges denote true assignments, and dashed edges denote false assignments. The `WMC` is computed using a single bottom-up pass where each node is annotated with its weighted model count. The `WMC` at each node is computed as the weighted sum of the `WMC` of its children. For instance, the `WMC` at node x_0 is computed as $0.5 \times 0.5 + 0.5 \times 0.5$. It is well known that BDDs exploit program structure and, in practice, can scale to large probabilistic inference tasks [17, 29].

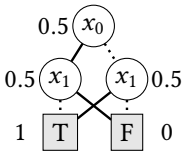


Fig. 7. Example BDD

4 Rosette: Formal Semantics

Section 5 gives a proof that Roulette's probabilistic adaptation of Rosette's symbolic evaluation strategy correctly implements probabilistic inference. Stating this correctness property first requires a formal model of Rosette-style symbolic evaluation to build upon. However, no existing formalism of Rosette contains a description of its symbolic evaluation strategy that is complete enough to serve as a foundation for Roulette's correctness theorem. Torlak and Bodík [61] give only sketches of an operational semantics and correctness proof. Porncharoenwase et al. [51] give a mechanized semantics, but it does not account for programs that dynamically generate symbolic variables or mutable references.

This section presents a model of Rosette that differs from prior work in a few ways. On the one hand, it supports dynamically generated symbolic values and mutable references, and the main theorem establishes correctness of Rosette-style symbolic evaluation in the presence of these features. On the other hand, it has a coarse-grained model of symbolic values and state-merging, and does not account for other features of Rosette such as `assume` statements and queries to the solver. This tradeoff is sensible given that the precise implementation details of symbolic state-merging and other solver-aided features of Rosette are not as relevant for Roulette, while dynamically generated symbolic values and mutable references are crucial for explaining the correctness of Roulette on the examples in Section 2.

The structure of this section parallels prior formalisms of Rosette. Section 4.1 presents a Scheme-like λ-calculus of idealized Rosette programs. We then give two operational semantics for it: a *concrete* semantics that defines standard call-by-value evaluation (Section 4.2) and an *abstract* semantics that defines the behavior of the symbolic evaluator (Section 4.3). Correctness of symbolic evaluation boils down to a *soundness and completeness* theorem linking these two semantics (Theorem 4.1). Soundness states that every concrete outcome of a given program is covered by its abstract semantics, and completeness states that every possible outcome covered by the abstract semantics of a program is reachable by concrete execution.

4.1 Syntax of Idealized Rosette

Figure 8 contains the syntax of idealized Rosette. On the left is the surface syntax, i.e., the syntax a programmer would write down. On the right is the evaluation syntax, i.e., the syntax needed solely to define an evaluation function. Idealized Rosette supports many standard features from functional languages: λ-expressions, conditionals, ML-style mutable references, pairs, arithmetic operations, and errors (`fail`).

There is one feature unique to symbolic evaluation: `sym`, a language form that models Rosette's `define-symbolic*`. This language form is treated differently in the concrete and abstract semantics. In the concrete semantics, programs are run with an infinite stream of Booleans that supplies a concrete Boolean in place of each `sym`. This is similar to the semantics of `random` for a pseudorandom number generator, where the return value comes from some infinite sequence of random numbers determined by a seed. In the abstract semantics, programs are not run with a stream but are instead *lifted* to operate on *symbolic values*, and `sym` generates a fresh symbolic value of Boolean type.

4.2 Concrete Semantics of Idealized Rosette

Figure 9 displays the environment-based natural semantics [31] for concrete execution. In addition to the usual environment ρ and store σ, the evaluation judgment also includes a stream s. The first element of s is used to evaluate `sym` expressions via the rules SymTrue and SymFalse. The remaining rules define a standard call-by-value semantics that additionally threads the stream throughout. Selected rules are shown in Figure 9. When encountering a syntactic form with two subexpressions

Surface Syntax

$$
\begin{aligned}
e \in \text{Expr} &::= x \mid \texttt{let}\ x = e\ \texttt{in}\ e \mid \lambda x.e \mid x\ y \\
&\mid b \mid \texttt{if}\ x\ e\ e \mid r \mid x + y \mid x - y \mid x \times y \\
&\mid () \mid (x, y) \mid \texttt{fst}\, x \mid \texttt{snd}\, x \\
&\mid \texttt{ref}\, x \mid \texttt{!}\, y \mid x := y \mid \texttt{sym} \mid \texttt{fail} \\
b \in \text{Bool} &::= \texttt{true} \mid \texttt{false} \\
r \in \text{Num} &::= \mathbb{Q} \\
x, y \in \text{Var}
\end{aligned}
$$

Evaluation Syntax

$$
\begin{aligned}
v, w \in \text{Val} &::= b \mid r \mid () \mid (v, w) \mid c \mid \ell \\
c \in \text{Closure} &::= \texttt{clo}(\lambda x.e, \rho) \\
\rho \in \text{Env} &::= \text{Var} \rightharpoonup_{\text{fin}} \text{Val} \\
\sigma \in \text{Store} &::= \text{Loc} \rightharpoonup_{\text{fin}} \text{Val} \\
s \in \text{Stream} &::= \mathbb{N} \to \mathbb{B} \quad \text{where } \mathbb{B} = \{\mathsf{T}, \mathsf{F}\} \\
\ell \in \text{Loc}
\end{aligned}
$$

Fig. 8. Syntax of Idealized Rosette

$\boxed{\rho, s \vdash (e, \sigma) \Downarrow (v, \sigma')}$

$$
\textsc{Let}\ \frac{s \rightsquigarrow s_1, s_2 \qquad \rho, s_1 \vdash (e_1, \sigma) \Downarrow (v_1, \sigma_1) \qquad \rho[x \mapsto v_1], s_2 \vdash (e_2, \sigma_1) \Downarrow (v_2, \sigma_2)}{\rho, s \vdash (\texttt{let}\ x = e_1\ \texttt{in}\ e_2, \sigma) \Downarrow (v_2, \sigma_2)}
$$

$$
\textsc{App}\ \frac{\rho(x) = \texttt{clo}(\lambda x'.\, e', \rho') \qquad \rho'[x' \mapsto \rho(y)], s \vdash (e', \sigma) \Downarrow (v, \sigma')}{\rho, s \vdash (x\ y, \sigma) \Downarrow (v, \sigma')}
$$

$$
\textsc{Ref}\ \frac{\ell \notin \text{locs}(\rho, \sigma)}{\rho, s \vdash (\texttt{ref}\, x, \sigma) \Downarrow (\ell, \sigma[\ell \mapsto \rho(x)])}
$$

$$
\textsc{Set}\ \frac{\rho(x) \in \text{dom}(\sigma)}{\rho, s \vdash (x := y, \sigma) \Downarrow ((), \sigma[\rho(x) \mapsto \rho(y)])}
$$

$$
\textsc{Get}\ \frac{\rho(x) \in \text{dom}(\sigma)}{\rho, s \vdash (!x, \sigma) \Downarrow (\sigma(\rho(x)), \sigma)} \qquad \textsc{SymTrue}\ \frac{}{\rho, \mathsf{T} :: s \vdash (\texttt{sym}, \sigma) \Downarrow (\texttt{true}, \sigma)} \qquad \textsc{SymFalse}\ \frac{}{\rho, \mathsf{F} :: s \vdash (\texttt{sym}, \sigma) \Downarrow (\texttt{false}, \sigma)}
$$

Fig. 9. Concrete Semantics of Idealized Rosette (Selected Rules)

(e.g., `let`), the stream is split into two independent pieces via the judgment $s \rightsquigarrow s_1, s_2$, which holds if and only if s_1 contains the even-indexed elements of s and s_2 contains the odd-indexed elements. Note that there is no rule for `fail`, so all traces that encounter `fail` are pruned from the concrete execution. The remaining rules are standard; for details, see Appendix B.

4.3 Abstract Semantics of Idealized Rosette

Defining the abstract semantics of Rosette involves lifting all the evaluation syntax to its symbolic counterpart. For example, values v are lifted to *symbolic values* $\widehat{v}$. Specifying this lifting requires notions of symbolic variables, symbolic propositions, and symbolic unions.

A *symbolic variable* is a Boolean variable that gives meaning to symbolic values generated by `sym`. Unlike ordinary Booleans, a symbolic variable can stand in for either T or F. Assuming the existence of an infinite supply of symbolic variables $\alpha, \beta, \cdots \in \text{SymVar}$, a *model* is an assignment of Boolean values to each symbolic variable. Let $\text{Model} = \text{SymVar} \to \mathbb{B}$ be the set of models. For any set of symbolic variables S, let $\text{Model}_S = S \to \mathbb{B}$ be the set of models over the variables in S. Symbolic execution allocates a finite set of symbolic variables S and produces a *symbolic value*, which is a function $\text{Model}_S \to \text{Val}$. More generally, the *symbolic elements of type* A, written $\widehat{A}$, is the set of functions $\text{Model} \to A_\perp$ where $A_\perp = A \uplus \{\perp\}$. For example, symbolic environments are written $\widehat{\rho} \in \widehat{\text{Env}}$. The *support* of a symbolic element $\widehat{a} \in \widehat{A}$, denoted $\text{symvars}(\widehat{a})$, is the smallest subset S of SymVar such that $\widehat{a}$ restricted to the domain $\text{Model}_{\text{symvars}(\widehat{a})}$ is equivalent to $\widehat{a}$.

$\boxed{\widehat{\rho} \vdash (e, \widehat{\sigma}) \Downarrow (\widehat{v}, \widehat{\sigma}', \psi)}$

$$\textsc{Let}\ \frac{\widehat{\rho} \vdash (e_1, \widehat{\sigma}) \Downarrow (\widehat{v}_1, \widehat{\sigma}_1, \psi_1) \quad \widehat{\rho}[x \mapsto \widehat{v}_1] \vdash (e_2, \widehat{\sigma}_1) \Downarrow (\widehat{v}_2, \widehat{\sigma}_2, \psi_2)}{\widehat{\rho} \vdash (\texttt{let}\ x = e_1\ \texttt{in}\ e_2, \widehat{\sigma}) \Downarrow ([\psi_1 : \widehat{v}_2], [\psi_1 : \widehat{\sigma}_2], \psi_1 \wedge \psi_2)}$$

$$\textsc{App}\ \frac{\widehat{\rho}(x) = [\varphi_i : \texttt{clo}(\lambda x_i.e_i, \widehat{\rho}_i)]_{i \in I} \uplus_{\text{Closure}} \widehat{v} \quad \forall i \in I.\ \widehat{\rho}_i[x_i \mapsto \widehat{\rho}(y)] \vdash (e_i, \widehat{\sigma}) \Downarrow (\widehat{v}_i, \widehat{\sigma}_i, \psi_i)}{\widehat{\rho} \vdash (x\ y, \widehat{\sigma}) \Downarrow ([\varphi_i : \widehat{v}_i]_{i \in I}, [\varphi_i : \widehat{\sigma}_i]_{i \in I}, \bigvee_i (\varphi_i \wedge \psi_i))}$$

$$\textsc{If}\ \frac{\widehat{\rho}(x) = [\varphi_1 : \texttt{true}, \varphi_2 : \texttt{false}] \uplus_{\text{Bool}} \widehat{v} \quad \widehat{\rho} \vdash (e_1, \widehat{\sigma}) \Downarrow (\widehat{v}_1, \widehat{\sigma}_1, \psi_1) \quad \widehat{\rho} \vdash (e_2, \widehat{\sigma}) \Downarrow (\widehat{v}_2, \widehat{\sigma}_2, \psi_2)}{\widehat{\rho} \vdash (\texttt{if}\ x\ e_1\ e_2, \widehat{\sigma}) \Downarrow ([\varphi_1 : \widehat{v}_1, \varphi_2 : \widehat{v}_2], [\varphi_1 : \widehat{\sigma}_1, \varphi_2 : \widehat{\sigma}_2], (\varphi_1 \wedge \psi_1) \vee (\varphi_2 \wedge \psi_2))}$$

$$\textsc{Fail}\ \frac{}{\widehat{\rho} \vdash (\texttt{fail}, \widehat{\sigma}) \Downarrow (\varnothing, \varnothing, \mathsf{F})} \qquad \textsc{Ref}\ \frac{\ell \text{ smallest not in } \text{locs}(\widehat{\rho}, \widehat{\sigma})}{\widehat{\rho} \vdash (\texttt{ref}\ x, \widehat{\sigma}) \Downarrow ([\mathsf{T} : \ell], \widehat{\sigma}[\ell \mapsto \widehat{\rho}(x)], \mathsf{T})}$$

$$\textsc{Get}\ \frac{\widehat{\rho}(x) = [\varphi_i : \ell_i]_{i \in I} \uplus_{\text{Loc}} \widehat{v}}{\widehat{\rho} \vdash (!x, \widehat{\sigma}) \Downarrow ([\varphi_i : \widehat{\sigma}(\ell_i)]_{i \in I}, \widehat{\sigma}, \bigvee_i \varphi_i)}$$

$$\textsc{Set}\ \frac{\widehat{\rho}(x) = [\varphi_i : \ell_i]_{i \in I} \uplus_{\text{Loc}} \widehat{v}}{\widehat{\rho} \vdash (x := y, \widehat{\sigma}) \Downarrow ([(\bigvee_i \varphi_i) : ()], [\varphi_i : \widehat{\sigma}[\ell_i \mapsto \widehat{\rho}(y)]]_{i \in I}, \bigvee_i \varphi_i)}$$

$$\textsc{Sym}\ \frac{\alpha \text{ smallest not in } \text{symvars}(\widehat{\rho}, \widehat{\sigma})}{\widehat{\rho} \vdash (\texttt{sym}, \widehat{\sigma}) \Downarrow ([\alpha : \texttt{true}, \neg\alpha : \texttt{false}], \widehat{\sigma}, \mathsf{T})}$$

Fig. 10. Abstract Semantics of Idealized Rosette (Selected Rules)

A *symbolic proposition* is an element of the set $\varphi, \psi \in \text{Model} \to \mathbb{B}$. Two symbolic propositions φ, ψ are *disjoint* if and only if $\varphi(m) \wedge \psi(m) = \mathsf{F}$ for all $m \in \text{Model}$. For a finite family $(\varphi_i)_{i \in I}$ of pairwise-disjoint symbolic propositions and a finite family of symbolic elements $(\widehat{a}_i)_{i \in I}$, their *symbolic union* $[\varphi_i : \widehat{a}_i]_{i \in I} \in \widehat{A}$ is the symbolic proposition defined by $([\varphi_i : \widehat{a}_i]_{i \in I})(m) = \widehat{a}_i(m)$ when $\varphi_i(m) = \mathsf{T}$ for some $i \in I$, and $([\varphi_i : \widehat{a}_i]_{i \in I})(m) = \bot$ when $\varphi_i(m) = \mathsf{F}$ for all $i \in I$.

Symbolic unions are functions out of the space of models, so symbolic quantities are automatically equal up to "nesting" of symbolic unions. This model of symbolic unions has been chosen specifically to facilitate reasoning about Rosette's symbolic evaluation strategy while abstracting over particular details of its implementation. Other Rosette models also abstract over the details of symbolic unions [51].

Figure 10 displays selected rules for defining the abstract semantics. As alluded to earlier, the abstract semantics lifts the input environment ρ and input store σ from the concrete semantics to a symbolic environment $\widehat{\rho}$ and symbolic store $\widehat{\sigma}$; accordingly, it produces symbolic values $\widehat{v}$. The rules are mostly analogous to the concrete ones, but there are a few important differences. Most notably, the $\Downarrow$ relation does not include a stream. This difference is due to the fundamental distinction between concrete and abstract execution: a symbolic evaluator computes *all* branches of execution simultaneously while a concrete evaluator does not.

For instance, consider the rule Sym. Instead of producing a particular Boolean value, the abstract semantics of `sym` generates a fresh symbolic variable and produces a symbolic union whose value is determined by the symbolic variable.

Also note the difference in the semantics of `if`, defined by the If rule. When encountering an `if`, the abstract semantics first evaluates the guard expression and determines the conditions φ_1 and φ_2 under which the guard expression produces the values `true` or `false`, respectively. The operator $\uplus_{\text{Bool}}$ indicates that the remainder of the symbolic union $\widehat{v}$ does not contain any Booleans—formally, $\text{img}(\widehat{v}) \cap \text{Bool} = \varnothing$. Both branches are evaluated, and the result is placed under a symbolic union with the corresponding guards. An additional output component ψ encodes the *assertion proposition*. This formula specifies models under which the output of the abstract semantics is defined. So, in models where the guard is not a Boolean, i.e., those mapped by $\widehat{v}$, the assertion proposition is not satisfied. The remaining rules can be found in Appendix B.

Answers

$$a \in \mathsf{Ans} ::= () \mid b \mid r \mid (a, a) \mid \mathsf{opaque}$$

$\lfloor \cdot \rfloor : \mathsf{Value} \to \mathsf{Ans}$

$$\lfloor v \rfloor = \begin{cases} v & \text{if } v \in \mathsf{Bool} \cup \mathsf{Num} \cup \{()\} \\ (\lfloor v_1 \rfloor, \lfloor v_2 \rfloor) & \text{if } v = (v_1, v_2) \\ \mathsf{opaque} & \text{otherwise} \end{cases}$$

$\mathsf{eval} : \mathsf{Expr} \to (\mathsf{Stream} \to \mathsf{Ans}_\perp)$

$$\mathsf{eval}(e)(s) = \begin{cases} \lfloor v \rfloor & \text{if } \varnothing, s \vdash (e, \varnothing) \Downarrow (v, \sigma) \\ \perp & \text{otherwise} \end{cases}$$

$\mathsf{ans} : \widehat{\mathsf{Value}} \times \widehat{\mathsf{Store}} \to \widehat{\mathsf{Ans}}$

$$\mathsf{ans}(\widehat{v}, \widehat{\sigma})(m) = \begin{cases} \lfloor \widehat{v}(m) \rfloor & \text{if } \widehat{v}(m) \neq \perp, \widehat{\sigma}(m) \neq \perp \\ \perp & \text{otherwise} \end{cases}$$

Fig. 11. Idealized Rosette Evaluation Functions

4.4 Correctness

The primary correctness property of a precise symbolic evaluator is that it produces a representation of all and only those results that can be produced by concrete execution. In idealized Rosette, *answers* are the result of computation. The left side of Figure 11 explicitly defines the set of answers Ans and the function $\lfloor \cdot \rfloor$ for converting values to answers.

The right side of Figure 11 defines the meaning of concrete and abstract execution in terms of answers. Given a closed term e, the function $\mathsf{eval}(e)$ says which answers can be feasibly produced by e via concrete execution. Given an abstract value $\widehat{v}$ and an abstract store $\widehat{\sigma}$ produced by the abstract execution of e, the function $\mathsf{ans}(\widehat{v}, \widehat{\sigma})$ says which answers abstract execution considers feasible. Theorem 4.1 shows that these two results coincide and additionally that the assertion proposition produced by abstract execution is true exactly when concrete execution produces a well-defined value.

Theorem 4.1 (Correctness of Idealized Rosette)**.** Let e be a closed term where $\varnothing \vdash (e, \varnothing) \Downarrow\!\Downarrow (\widehat{v}, \widehat{\sigma}, \psi)$. Then $\mathsf{img}(\mathsf{eval}(e)) = \mathsf{img}(\mathsf{ans}(\widehat{v}, \widehat{\sigma}))$, and for all $m \in \mathsf{Model}_{\mathsf{symvars}(\widehat{v}, \widehat{\sigma})}$ it holds that $\psi(m) = \mathsf{T}$ if and only if $\widehat{v}(m) \neq \perp$ and $\widehat{\sigma}(m) \neq \perp$.

Proof sketch. The proof is by induction, after generalizing the theorem statement to open terms. To systematically work up to renaming of store locations, the proof contains a nontrivial use of nominal techniques [49]. For details, see Appendix B. □

5 Roulette: Formal Semantics

Given the semantics of Rosette, only a few small changes are needed to model Roulette. Specifically, each symbolic variable in the abstract semantics is additionally associated with a *weight*, which intuitively corresponds to the probability of that variable taking on the value T. Everything else can be adapted in a straightforward way. This section follows the same structure as the previous one, showing only the changes needed to turn Rosette into Roulette.

5.1 Syntax of Roulette

In the syntax of Roulette, the `sym` expression for generating symbolic Booleans is replaced by the expression `flip` x for generating a random Boolean with bias x. For concrete execution, the value of `flip` x is determined by the first element in a stream of real numbers $s \in \mathsf{Stream} \coloneqq \mathbb{N} \to [0, 1]$. This new stream, also known as an entropy source [11], provides a pool of uniform randomness for sampling. For abstract execution, `flip` returns a symbolic variable that is associated with a weight. This association is stored in a *weight map* $w \in \mathsf{WeightMap} \coloneqq \mathsf{SymVar} \to_{\mathsf{fin}} [0, 1]$.

$\boxed{\rho, s \vdash (e, \sigma) \Downarrow (v, \sigma')}$

$$\textsc{FlipTrue}\ \frac{x \in \mathrm{dom}(\rho) \qquad r < \rho(x)}{\rho, r :: s \vdash (\mathtt{flip}\, x, \sigma) \Downarrow (\mathtt{true}, \sigma)} \qquad \textsc{FlipFalse}\ \frac{x \in \mathrm{dom}(\rho) \qquad r \geq \rho(x)}{\rho, r :: s \vdash (\mathtt{flip}\, x, \sigma) \Downarrow (\mathtt{false}, \sigma)}$$

Fig. 12. Concrete Semantics

$\boxed{\widehat{\rho} \vdash (e, \widehat{\sigma}, w) \Downarrow\!\Downarrow (\widehat{v}, \widehat{\sigma}', w', \psi)}$

$$\textsc{Flip}\ \frac{\widehat{\rho}(x) = [\varphi_i : r_i]_{1 \leq i \leq n} \uplus_{\mathrm{Num}} \widehat{v} \qquad \forall 1 \leq i \leq n.\ s_i = \max(\min(r_i, 1), 0) \qquad \alpha_1, \ldots, \alpha_n \text{ smallest not in } \mathrm{dom}(w)}{\widehat{\rho} \vdash (\mathtt{flip}\, x, \widehat{\sigma}, w) \Downarrow\!\Downarrow ([\varphi_i \wedge \alpha_i \mapsto \mathsf{T},\ \varphi_i \wedge \neg\alpha_i \mapsto \mathsf{F}]_{i \in I},\ \widehat{\sigma},\ w \uplus \{\alpha_i \mapsto s_i\}_{i \in I},\ \bigvee_i \varphi_i)}$$

Fig. 13. Abstract Semantics

5.2 Concrete Semantics of Roulette

The concrete semantics of Roulette is given in Figure 12. There are two rules here intended to replace SymTrue and SymFalse. The `flip` form uses the first element of the entropy source to determine if `true` or `false` should be returned, based on the probability given to `flip`. This is exactly how `flip` is implemented in a sampling semantics where the entropy determines the values returned by the random-number generator.

5.3 Abstract Semantics of Roulette

The abstract semantics of Roulette is given in Figure 13. At a high-level, the Flip rule in the abstract semantics returns a fresh symbolic variable and a new weight map that associates the new variable with its probability. If the input probability is a concrete number, then only one symbolic variable is generated. Supporting symbolic arguments to `flip` takes some extra work and requires generating more symbolic variables. For each number in the symbolic union given to `flip`, the semantics generates a fresh symbolic variable and associates it with the given number. Since each r_i can be an arbitrary rational number, not just a probability, every r_i must be clamped to the interval. As in idealized Rosette, this rule is made deterministic by requiring $\alpha_1, \ldots, \alpha_n$ to be the smallest sequence of symbolic variables that is fresh for the variables currently in use.

5.4 Correctness

Theorem 4.1 generalizes smoothly from Rosette to Roulette. For Rosette, correctness amounted to showing that the abstract and concrete semantics compute the same set of possible answers. For Roulette, it is not only the set of possible answers that must be identical, but also the probability of each answer.

More precisely, the correctness theorem for Rosette required $\mathrm{eval}(e) : \mathrm{Stream} \to \mathrm{Ans}_\perp$ and $\mathrm{ans}(\widehat{v}, \widehat{\sigma}) : \mathrm{Model}_V \to \mathrm{Ans}_\perp$ to have the same image. In the analogous theorem for Roulette, these functions become *random variables* out of the sample spaces $\mathrm{Stream} = \mathbb{N} \to [0, 1]$ and $\mathrm{Model}_{\mathrm{dom}(\omega)}$, respectively, where ω is the weight map assigning probabilities to each symbolic variable generated during symbolic evaluation. These sample spaces naturally come with probability measures: Stream comes with the uniform Lebesgue measure following Culpepper and Cobb [11], and $\mathrm{Model}_{\mathrm{dom}(\omega)}$ comes with the measure that assigns a given model $m \in \mathrm{Model}_{\mathrm{dom}(\omega)}$ its *weight*: $\mathrm{weight}_w(m) = \prod_{(x \mapsto b) \in m} (\text{if } b \text{ then } w(x) \text{ else } 1 - w(x))$. For example, under the weight map

$w = [\alpha_0 \mapsto 0.1, \alpha_1 \mapsto 0.2]$, the model $m = [\alpha_0 \mapsto \mathsf{T}, \alpha_1 \mapsto \mathsf{F}]$ has $\mathsf{weight}_w(m) = 0.1 \cdot 0.8 = 0.08$. Correctness of Roulette states that the random variables $\mathsf{eval}(e)$ and $\mathsf{ans}(\widehat{v}, \widehat{\sigma})$ have the same distribution under these probability measures, and that the assertion proposition captures when these random variables are not equal to $\bot$. The proof follows the same structure as Theorem 4.1.

Theorem 5.1 (Correctness of Roulette)**.** Let e be a closed term such that $\varnothing \vdash (e, \varnothing, \varnothing) \Downarrow (\widehat{v}, \widehat{\sigma}, w, \psi)$. Then the random variables $\mathsf{eval}(e) : \mathsf{Stream} \to \mathsf{Ans}_\bot$ and $\mathsf{ans}(\widehat{v}, \widehat{\sigma}) : \mathsf{Model}_{\mathsf{dom}(w)} \to \mathsf{Ans}_\bot$ have the same distribution, where the entropy source Stream is given the uniform Lebesgue measure and $\mathsf{Model}_{\mathsf{dom}(w)}$ is given the measure weight_w. Moreover, for all $m \in \mathsf{Model}_{\mathsf{dom}(w)}$ it holds that $\psi(m) = \mathsf{T}$ if and only if $\widehat{v}(m) \neq \bot$ and $\widehat{\sigma}(m) \neq \bot$.

Proof. The proof is by induction, after generalizing to open terms. For details, see Appendix C. □

6 Evaluation

The examples in Section 2 demonstrate that Roulette is expressive, but to be useful, a probabilistic programming language must support *efficient* inference. This section confirms that Roulette is capable of scaling better than enumeration, and in general its performance is competitive with Dice [29], a state-of-the-art PPL for performing exact discrete inference. Unlike Roulette, Dice is extremely restricted: it has no support for recursion, mutable state, or many of the other features supported by Roulette.

The evaluation consists of three sets of programs. First is a series of small baseline programs commonly used to test PPLs in the literature. These programs are small enough that they finish nearly instantly. Second is a selection of discrete Bayesian networks intended to exercise the limits of efficient inference. These benchmarks come from the Bayesian Network Repository, a collection of Bayesian networks taken from real-world scenarios. All the networks are run on a program similar to the one described in Section 2.2. The final set of programs measures how well Roulette scales on the unreliable hardware example from Section 2.3, hidden Markov model queries, and the scaling experiments from Holtzen et al. [29].

Experimental Setup. We ran the experiments on a 16-core AMD EPYC 7543 CPU with 128GB of RAM using Racket 8.13 and Rosette 4.1. The benchmarks compare against two versions of Dice: a recent build (Git commit `ed86716`), referred to as Dice (2025), and the original version from Holtzen et al. [29], referred to as Dice (2020). All experiments were performed on a best-effort basis where an attempt was made to find the optimal encoding for the program in each language. Additionally, BDD performance is significantly impacted by variable order, and changing the variable order can increase the size of a BDD from linear to exponential [34]. To the extent possible, the experiments control for variable order: both Dice and Roulette use the program order heuristic (i.e., random variables generated earlier during execution come earlier in the variable order) and the programs are written such that the variable order is consistent between each system.

6.1 Results

The main aim in these experiments is to establish that Roulette is competitive with existing high-performance exact inference strategies while gaining expressivity. In general, there should be no expectation that Roulette significantly outperforms existing knowledge-compilation-based PPLs such as Dice. However, Roulette should also not exhibit a significant performance penalty as a price for its expanded expressivity. The remainder of this section presents the experimental results in brief and states their consequences. Then, Section 6.2 gives a higher-level discussion about performance that examines cross-cutting trends across the experiments. Some experimental results are truncated in the main body; see Appendix D for the full experimental results.

Table 1. Simple Baselines

	Roulette		Dice (2025)		Dice (2020)		PSI
Benchmark	BDD Size	Time (ms)	BDD Size	Time (ms)	BDD Size	Time (ms)	Time (ms)
ALARM	**11**	**0**	**11**	19	**11**	16	71
EVIDENCE1	**3**	**0**	5	24	5	17	22
EVIDENCE2	**4**	**0**	6	21	6	18	34
GRASS	**13**	**0**	15	30	15	19	80
MURDER-MYSTERY	**4**	**0**	6	28	6	20	46
NOISY-OR	**33**	**0**	35	23	35	19	328
TWO-COINS	**3**	**0**	5	25	5	22	20

Table 2. Bayesian Networks

	Roulette		Dice (2025)		Dice (2020)	
Benchmark	BDD Size	Time (ms)	BDD Size	Time (ms)	BDD Size	Time (ms)
CANCER	**13**	**2**	15	49	28	19
SURVEY	**46**	**4**	48	29	73	18
ALARM	981	42	**672**	308	1,366	**30**
INSURANCE	75,594	395	**44,846**	643	101,047	**148**
HEPAR2	**1,967**	140	1,969	230	3,936	**32**
HAILFINDER	**33,211**	596	–	–	65,386	**428**
PIGS	**19**	255	25	417	35	**48**
WATER	39,146	**200**	**33,226**	454	51,952	16,083
MUNIN	**10,307**	2,400	3,704	24,839	11,977	**1,605**

6.1.1 Simple baselines. Table 1 shows the results of the benchmarks introduced by Gehr et al. [23]. The time taken by Roulette and Dice to solve these problems is nearly instant in all cases. PSI [23] is included as a comparison point in this experiment, which lags behind the performance of the knowledge-compilation-based approaches. The table additionally reports the BDD size (in number of nodes) to show the degree to which the different encoding schemes of Roulette, Dice (2020), and Dice (2025) impact performance. Roulette achieves the smallest BDD in all cases.

6.1.2 Bayesian networks. Table 2 shows the results for single-marginal inference on the nine discrete Bayesian networks evaluated in the original Dice paper [29]. The "–" symbol indicates a timeout after one minute. Single-marginal inference calculates the marginal probability for a single leaf node in a network. These results are for a Bayesian network interpreter that is almost identical to the one in Section 2.2.

Overall, Roulette is competitive with both versions of Dice on these examples, with similar overall BDD size and execution time. In the table, networks are roughly ordered by difficulty:

- *Small networks.* CANCER and SURVEY are small and therefore inference is relatively easy. Roulette has the best performance on these.
- *Medium networks.* ALARM, INSURANCE, HEPAR2, and HAILFINDER are medium-sized Bayesian networks. In general, Dice (2020) has the smallest execution time on these networks, but Roulette is close in performance. Roulette compiles to the smallest final BDD in most of these examples, suggesting that its encoding is quite efficient. There were some performance regressions in Dice (2025), and it failed to complete HAILFINDER.
- *Large networks.* PIGS, WATER, and MUNIN are large and challenging networks with many thousands of parameters. Roulette is significantly faster than both versions of Dice on the WATER network. On the MUNIN network, Roulette generates the smallest BDD, but its performance is marginally inferior to Dice (2020) and lags by about 800 milliseconds.

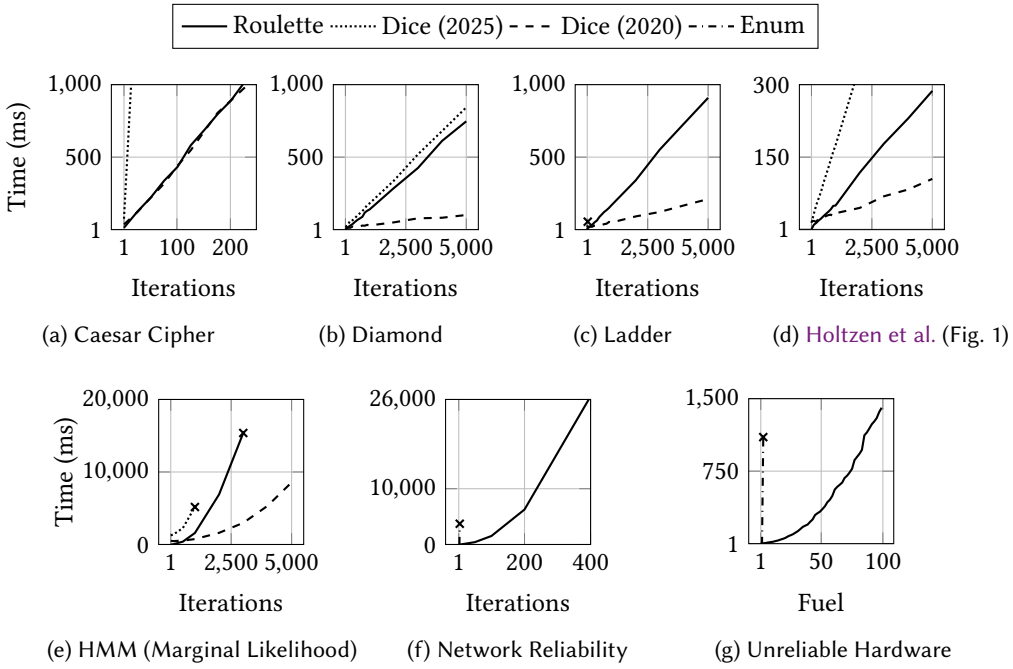


Fig. 14. Plots (a), (b), (c), and (d) parallel Figure 10 from Holtzen et al. [29]. Plot (e) is a hidden Markov model example. These five plots compare against Dice. Plot (f) is the network reliability program from Section 2.1, and Plot (g) is the unreliable hardware program from Section 2.3. These benchmarks compare against enumeration. An × shows the last trial to finish within a one-minute timeout.

Overall, these experiments show that Roulette is not giving up significant performance to attain its expressivity. Roulette can complete all the Bayesian network tasks within a reasonable amount of time and in some cases outperforms Dice on challenging examples such as WATER.

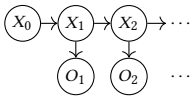
6.1.3 Scaling benchmarks. Figure 14 shows the results of experiments that examine how Roulette and Dice scale as parameters grow in size. Some of these plots reproduce benchmarks from the original performance evaluation of Dice, while other benchmarks are new. Here is a detailed analysis of these experiments:

- Figure 14a is the Caesar cipher example from Figure 3 of Holtzen et al. [29]. The program performs frequency analysis to decrypt text that is encrypted using a Caesar cipher. The scaling parameter is the length of the ciphertext. Roulette and Dice (2020) achieve nearly identical performance on this example.
- Figure 14b and Figure 14c examine the probabilistic network reliability benchmarks from Holtzen et al. [29]. The following shows diamond and ladder on the left and right, respectively:

The scaling parameter is the number of repeated subnetworks (boxed). The task is a simplified version of the example from Figure 1: for a particular network topology where each edge has some chance of dropping a packet and each router forwards uniformly at random, determine whether a packet successfully traverses the network. Here, Roulette and Dice (2020) both scale

linearly, with Dice (2020) having overall superior constant factors. Dice (2025) scales surprisingly poorly on ladder despite using the same BDD backend as Roulette. Overall, these experiments demonstrate that Roulette and Dice (2020) have similar asymptotic behavior, with Dice (2020) having superior constant factors. This performance difference is not likely to be an inherent limitation of Roulette but is more likely due to confounding factors, such as Dice (2020) using a different BDD backend.

- Figure 14d is the simple Markov chain example from Holtzen et al. [29, Figure 1]. Again, all tools achieve linear-time performance with Dice (2020) having superior constant factors.
- Figure 14e shows the scaling behavior for querying the following hidden Markov model (HMM):



  The scaling parameter is the length of the HMM. The inference task is, for an HMM of length n, to compute the probability that all the O_i are true (each node in the network is a Boolean random variable). In addition to these automated approaches (Roulette and Dice), we implemented a specialized inference algorithm that performs linear-time exact inference on HMMs [52]. These results are detailed in Appendix D, along with the performance of several other HMM queries. The automated approaches exhibit super-linear scaling behavior, with Dice (2020) having the best performance. Scaling is quadratic for Roulette and Dice due to the way that the BDD is constructed: each additional layer in an HMM requires traversing the entire previously-constructed (linear-size) BDD once, which scales quadratically in n.
- Finally, Figure 14f and Figure 14g examine how Roulette scales on some of the examples from Section 2. The baseline for comparison is enumeration since these programs are not easily representable in Dice. Figure 14f shows how the network reliability example from Section 2.1 scales in the `SIZE` parameter, and Figure 14g shows how the unreliable hardware example from Section 2.3 scales as `(tri (max 0 (- fuel 2)))` is evaluated on varying values of `fuel`. Enumeration times out quickly and exhibits exponential growth, while Roulette is able to scale comfortably. A comparison of least-squares regressions shows that, along all metrics, a quadratic model fits the data better than an exponential model for both the network reliability and unreliable hardware examples.

6.2 Performance Discussion

The experiments demonstrate that Roulette and Dice have similar scaling characteristics but exhibit constant-factor performance differences on some examples. While Roulette, Dice (2025), and Dice (2020) all use knowledge compilation for probabilistic inference, there are differences that impact the performance of these languages on the benchmarks. First, Roulette and Dice (2025) use the RSDD library [30] for managing BDDs, while Dice (2020) uses the CUDD library [58]. Second, Roulette is a DSL hosted in Racket, while Dice has a special-purpose compiler. As such, Dice generally has lower constant-time factors than Roulette. The Rosette runtime also performs formula simplification, caching, and other tasks that increase overhead.

Nonetheless, it is possible to achieve strong performance in Roulette by leveraging the host language's expressivity to control the way in which knowledge compilation is performed. This is similar to programming in Rosette, where changes to the structure of a program can have a significant influence on the resulting encoding and subsequently on the performance of queries to the underlying solver [50]. The expressivity afforded by Roulette allows developers to tune programs in this way. For example, a recursive function may be written in tail form or not, affecting the order in which variables are generated and hence affecting the compactness of BDD encoding.

Roulette's expressivity can help developers write optimized programs in the following ways:

- *Variable order.* The variable order in Roulette is controlled by the order in which random variables are allocated using `flip`. Hence, it is occasionally necessary for the programmer to pre-allocate random variables in order to satisfy some ordering constraint for optimal compilation.
- *Encoding strategy.* The way that data is represented as Boolean formulae is a critical component in the scalability of knowledge compilation strategies [5]. For instance, a number can be represented in binary or with a one-hot encoding. The choice of representation can greatly impact the performance of certain operations. Unlike Dice, Roulette is expressive enough to give this power to users. The Caesar cipher example in Figure 14a uses ordinary functional programming to implement a binary encoding of numbers, which is critical to being competitive with Dice.
- *Compilation order.* Roulette works by generating a Boolean formula that is shipped to a knowledge compiler. The performance of a knowledge compiler is sensitive to the formula it is given. Hence, control over formula structure is critical for achieving linear-time performance on the benchmarks shown in Figures 14b to 14d. In particular, it is necessary to compile the chain "from back to front." This strategy differs from the more naive implementation approach; see Appendix E for a concrete comparison of these two approaches.

These optimizations are available to programmers and can be considered "design patterns" for scalable inference. They follow the general guidance of aiming to (1) have related variables close to each other in the variable order and (2) compile related parts of the program together.

Beyond these guidelines there are additional Roulette-specific low-level optimizations, needed to obtain competitive performance with Dice, that are not available to the programmer for customization. Roulette has to work around some limitations of Rosette that do not occur in Dice. For instance, conditionals in Dice are compiled to `ite` operations on BDDs (which are more efficient), but Rosette does not have a Boolean-valued `ite` node in its formula representation. Therefore, places where conditionals occur have to be inferred, perhaps imperfectly, from the formulae.

Despite these optimizations, it is still the case that Dice outperforms Roulette on some examples. The remaining performance gap is likely due to a combination of (1) inherent overhead of Roulette being hosted in Racket, (2) differing knowledge compilation backends exhibiting different performance on the same examples, and (3) additional optimizations that are available in Dice due to the particularly constrained language.

7 Related Work

Probabilistic Programming Languages. The most closely related probabilistic programming languages to Roulette are those that rely on weighted model counting for exact inference such as Dice [29] and ProbLog [17]. Both Dice and ProbLog rely on a laborious and manual translation of the program structure into weighted Boolean formulae. Roulette vastly simplifies the implementation of these languages and increases expressivity, as demonstrated in Section 2. The exact performance of WMC-based inference is sensitive to how constraints are generated. As shown by Cao et al. [5], subtle choices about how to encode integers can result in different performance characteristics. It is somewhat surprising that Roulette performs as well as it does despite minimal manual optimization beyond what is provided by Rosette; this is a testament to the quality of the formula simplification provided already by the Rosette symbolic evaluator. Nonetheless, it would be interesting future work to integrate the optimizations explored by Cao et al. [5] into Roulette. Garg et al. [21] bit blast continuous random variables in order to use exact WMC-based inference for reasoning about continuous probability distributions; it would also be potentially fruitful to integrate these compilation strategies into Roulette.

While Roulette is expressive for a PPL, it nonetheless omits two useful features: continuous random variables and almost-sure termination. These features are found in many PPLs, but their presence makes exact inference challenging because they require integration. Hakaru [38] and PSI [23] support exact inference in the presence of continuous random variables, and their inference strategies consequently rely on expensive exact integration performed by computer algebra systems. SPPL [55] also supports exact inference in the presence of continuity, but it is a restricted language that lacks mutable state, higher-order functions, and general recursion. Approaches to handling almost-sure termination rely on fix-point solving [8] or generating functions [33]; it would be interesting to adapt these approaches to work with Roulette programs.

Solver-aided Programming. Solver-aided programming was introduced by Rosette [60, 61] and has since been applied in variety of domains. Rosette has been used to verify JIT compilers within the Linux kernel [39], develop a policy specification language for router configuration [64], create an instruction set architecture (ISA) emulator for building correct superoptimizers [48], and even verify safety properties of medical devices [47].

Porncharoenwase et al. [51] present a formally verified semantics of Rosette. Specifically, their $\mathcal{S}_c$ semantics is parameterized by a symbolic factory that abstracts the implementation details of symbolic values. Section 4 does not build on $\mathcal{S}_c$ primarily because it lacks support for dynamically generating symbolic variables and mutable state.

Symbolic Execution for Probabilistic Programming. Applying techniques in symbolic execution to probabilistic programming is not a new idea. Geldenhuys et al. [25] repurpose a symbolic execution framework for verifying Java programs to determine the probability that a function under test encounters a bug. This system relies on model counting, which weights all possible test inputs uniformly to determine the probability of failure and lacks Bayesian conditioning. Susag et al. [59] propose probabilistic symbolic variables for the Plinko language. Plinko is limited to finite loops, variable assignment, conditionals, and drawing symbolic variables from discrete probability distributions. Voogd et al. [62] present a full run-time semantics for probabilistic programming inspired by symbolic execution, called symProb, which performs bounded symbolic execution. However, symProb lacks higher-order functions and higher-order mutable state.

8 Conclusion

Roulette is an expressive discrete probabilistic programming language with exact inference built on top of Rosette. The key idea of Roulette is to adapt the symbolic compilation strategy of Rosette to generate symbolic constraints that can be given to a WMC-based inference backend. This strategy yields both an expressive and efficient probabilistic programming environment. To prove Roulette sound, we first gave a correctness theorem for an idealized version of Rosette that generalizes prior Rosette formalisms by including dynamically introduced symbolic values and mutable state. Idealized Rosette then served as the foundation for an analogous correctness theorem of Roulette. Finally, a suite of experiments showed that Roulette can achieve competitive performance on discrete exact inference for several challenging tasks.

In the future, we hope to make Roulette more usable and scalable by further connecting ideas from knowledge-compilation-based inference and Rosette. There are a variety of high-quality WMC solvers that would be interesting to explore as alternatives to BDDs as backends for Roulette [9, 12, 13, 35, 36]. Since Roulette is as expressive as Rosette, future work may also explore the extent to which the numerous existing Rosette examples benefit from a probabilistic interpretation. And finally, Rosette has benefited from many years of usability improvements, including symbolic profiling [3], performance repair [50], and typed lenient symbolic evaluation [6]. The extent to which these can be generalized and applied to Roulette is also exciting future work.

Acknowledgments

This work was supported by NSF grants SHF 2116372 and FMitF 2220408. The authors would like to thank Ryan Culpepper, Lisa Oakley, Sam Stites, and the anonymous PLDI reviewers, for their comments and suggestions.

Data Availability Statement

An up-to-date version of the software and appendices associated with this paper can be found at: https://doi.org/10.5281/zenodo.15049041.

References

[1] Michael Ballantyne, Mitch Gamburg, and Jason Hemann. 2024. Compiled, Extensible, Multi-language DSLs (Functional Pearl). In *International Conference on Functional Programming (ICFP)*. https://doi.org/10.1145/3674627

[2] Gilles Barthe, Boris Köpf, Federico Olmedo, and Santiago Zanella Beguelin. 2012. Probabilistic Relational Reasoning for Differential Privacy. In *Principles of Programming Languages (POPL)*. https://doi.org/10.1145/2492061

[3] James Bornholt and Emina Torlak. 2018. Finding Code That Explodes Under Symbolic Evaluation. In *Object-Oriented Programming, Systems, Languages and Applications (OOPSLA)*. https://doi.org/10.1145/3276519

[4] William E Byrd. 2009. *Relational Programming in miniKanren: Techniques, Applications, and Implementations*. Ph. D. Dissertation. Indiana University.

[5] William X Cao, Poorva Garg, Ryan Tjoa, Steven Holtzen, Todd Millstein, and Guy Van den Broeck. 2023. Scaling Integer Arithmetic in Probabilistic Programs. In *Uncertainty in Artificial Intelligence (UAI)*. https://doi.org/10.48550/arXiv.2307.13837

[6] Stephen Chang, Alex Knauth, and Emina Torlak. 2018. Symbolic Types for Lenient Symbolic Execution. In *Principles of Programming Languages (POPL)*. https://doi.org/10.1145/3158128

[7] Mark Chavira and Adnan Darwiche. 2008. On Probabilistic Inference by Weighted Model Counting. *Artificial Intelligence*. https://doi.org/10.1016/j.artint.2007.11.002

[8] David Chiang, Colin McDonald, and Chung-chieh Shan. 2023. Exact Recursive Probabilistic Programming. In *Object-Oriented Programming, Systems, Languages and Applications (OOPSLA)*. https://doi.org/10.1145/3586050

[9] Arthur Choi, Doga Kisa, and Adnan Darwiche. 2013. Compiling Probabilistic Graphical Models Using Sentential Decision Diagrams. In *Symbolic and Quantitative Approaches to Reasoning with Uncertainty*. https://doi.org/10.1007/978-3-642-39091-3_11

[10] Shumo Chu, Chenglong Wang, Konstantin Weitz, and Alvin Cheung. 2017. Cosette: An Automated Prover for SQL. In *Conference on Innovative Data Systems Research (CIDR)*.

[11] Ryan Culpepper and Andrew Cobb. 2017. Contextual Equivalence for Probabilistic Programs with Continuous Random Variables and Scoring. In *European Symposium on Programming (ESOP)*. https://doi.org/10.1007/978-3-662-54434-1_14

[12] Adnan Darwiche. 2002. A Compiler for Deterministic, Decomposable Negation Normal Form. In *AAAI Conference on Artificial Intelligence (AAAI)*.

[13] Adnan Darwiche. 2011. SDD: A New Canonical Representation of Propositional Knowledge Bases. In *International Joint Conference on Artificial Intelligence (IJCAI)*. https://doi.org/10.5591/978-1-57735-516-8/IJCAI11-143

[14] Adnan Darwiche and Pierre Marquis. 2002. A Knowledge Compilation Map. *Journal of Artificial Intelligence Research*. https://doi.org/10.1613/jair.989

[15] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An Efficient SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. https://doi.org/10.1007/978-3-540-78800-3_24

[16] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. 2007. Problog: A Probabilistic Prolog and Its Application in Link Discovery. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

[17] Daan Fierens, Guy Van den Broeck, Joris Renkens, Dimitar Shterionov, Bernd Gutmann, Ingo Thon, Gerda Janssens, and Luc De Raedt. 2015. Inference and Learning in Probabilistic Logic Programs Using Weighted Boolean Formulas. *Theory and Practice of Logic Programming*. https://doi.org/10.1017/S1471068414000076

[18] Matthew Flatt and PLT. 2010. *Reference: Racket*. Technical Report PLT-TR-2010-1. PLT Design Inc. https://racket-lang.org/tr1/.

[19] Daniel P. Friedman, William E. Byrd, Oleg Kiselyov, and Jason Hemann. 2018. *The Reasoned Schemer*. MIT Press.

[20] Norbert Fuhr. 1995. Probabilistic Datalog — A Logic for Powerful Retrieval Methods. In *Conference on Research and Development in Information Retrieval (SIGIR)*. https://doi.org/10.1145/215206.215372

[21] Poorva Garg, Steven Holtzen, Guy Van den Broeck, and Todd Millstein. 2024. Bit Blasting Probabilistic Programs. In *Programming Language Design and Implementation (PLDI)*. https://doi.org/10.1145/3656412

[22] Timon Gehr, Sasa Misailovic, Petar Tsankov, Laurent Vanbever, Pascal Wiesmann, and Martin Vechev. 2018. Bayonet: Probabilistic Inference for Networks. In *Programming Language Design and Implementation (PLDI)*. https://doi.org/10.1145/3296979.3192400

[23] Timon Gehr, Sasa Misailovic, and Martin Vechev. 2016. PSI: Exact Symbolic Inference for Probabilistic Programs. In *Computer Aided Verification (CAV)*. https://doi.org/10.1007/978-3-319-41528-4_4

[24] Timon Gehr, Samuel Steffen, and Martin T. Vechev. 2020. λPSI: Exact Inference for Higher-Order Probabilistic Programs. In *Programming Language Design and Implementation (PLDI)*. https://doi.org/10.1145/3385412.3386006

[25] Jaco Geldenhuys, Matthew B Dwyer, and Willem Visser. 2012. Probabilistic Symbolic Execution. In *International Symposium on Software Testing and Analysis (ISSTA)*. https://doi.org/10.1145/2338965.2336773

[26] Jason Hemann and Daniel P. Friedman. 2013. μKanren: A Minimal Functional Core for Relational Programming. In *Scheme and Functional Programming Workshop*.

[27] Jason Hemann, Daniel P. Friedman, William E. Byrd, and Matthew Might. 2017. A Simple Complete Search for Logic Programming. In *International Conference on Logic Programming (ICLP)*. https://doi.org/10.4230/OASICS.ICLP.2017.14

[28] Ralf Hinze and Colin Runciman. 2022. Super-Naturals. *Journal of Functional Programming (JFP)*. https://doi.org/10.1017/s0956796822000028

[29] Steven Holtzen, Guy Van den Broeck, and Todd Millstein. 2020. Scaling Exact Inference for Discrete Probabilistic Programs. In *Object-Oriented Programming, Systems, Languages and Applications (OOPSLA)*. https://doi.org/10.1145/3428208

[30] Steven Holtzen, Matthew Wang, John Gouwar, Sam Stites, and Minsung Cho. 2025. The Rust Decision Diagram Library (RSDD). https://github.com/neuppl/rsdd

[31] Gilles Kahn. 1987. Natural Semantics. In *Symposium on Theoretical Aspects of Computer Science (STACS)*. https://doi.org/10.1007/BFB0039592

[32] Rafael Kiesel and Thomas Eiter. 2023. Knowledge Compilation and More with SharpSAT-TD. In *Principles of Knowledge Representation and Reasoning (KR)*. https://doi.org/10.24963/kr.2023/40

[33] Lutz Klinkenberg, Christian Blumenthal, Mingshuai Chen, Darion Haase, and Joost-Pieter Katoen. 2024. Exact Bayesian Inference for Loopy Probabilistic Programs using Generating Functions. In *Object-Oriented Programming, Systems, Languages and Applications (OOPSLA)*. https://doi.org/10.1145/3649844

[34] Donald E Knuth. 2009. *The Art of Computer Programming*. Vol. 4. Addison-Wesley Professional.

[35] Jean-Marie Lagniez and Pierre Marquis. 2017. An Improved Decision-DNNF Compiler. In *International Joint Conference on Artificial Intelligence (IJCAI)*. https://doi.org/10.24963/ijcai.2017/93

[36] Christian Muise, Sheila McIlraith, J Christopher Beck, and Eric Hsu. 2012. Dsharp: Fast d-DNNF Compilation with sharpSAT. In *Advances in Artificial Intelligence*. https://doi.org/10.1007/978-3-642-30353-1_36

[37] Lawrence Murray, Daniel Lundén, Jan Kudlicka, David Broman, and Thomas Schön. 2018. Delayed Sampling and Automatic Rao-Blackwellization of Probabilistic Programs. In *International Conference on Artificial Intelligence and Statistics*.

[38] Praveen Narayanan, Jacques Carette, Wren Romano, Chung-chieh Shan, and Robert Zinkov. 2016. Probabilistic Inference by Program Transformation in Hakaru (System Description). In *Functional and Logic Programming (FLOPS)*. https://doi.org/10.1007/978-3-319-29604-3_5

[39] Luke Nelson, Jacob Van Geffen, Emina Torlak, and Xi Wang. 2020. Specification and Verification in the Field: Applying Formal Methods to BPF Just-in-Time Compilers in the Linux Kernel. In *Operating Systems Design and Implementation (OSDI)*.

[40] Raymond Ng and Venkatramanan Siva Subrahmanian. 1992. Probabilistic Logic Programming. *Information and Computation*. https://doi.org/10.1016/0890-5401(92)90061-J

[41] Lisa Oakley, Steven Holtzen, and Alina Oprea. 2024. Synthesizing Tight Privacy and Accuracy Bounds via Weighted Model Counting. In *Computer Security Foundations Symposium (CSF)*. https://doi.org/10.1109/CSF61375.2024.00048

[42] Scott Owens, Magnus O. Myreen, Ramana Kumar, and Yong Kiam Tan. 2016. Functional Big-Step Semantics. In *European Symposium on Programming (ESOP)*. https://doi.org/10.1007/978-3-662-49498-1_23

[43] Umut Oztok, Arthur Choi, and Adnan Darwiche. 2016. Solving PP^{PP}-Complete Problems using Knowledge Compilation. In *Principles of Knowledge Representation and Reasoning (KR)*.

[44] Umut Oztok and Adnan Darwiche. 2015. A Top-Down Compiler for Sentential Decision Diagrams. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

[45] Umut Oztok and Adnan Darwiche. 2018. An Exhaustive DPLL Algorithm for Model Counting. *Journal of Artificial Intelligence Research*. https://doi.org/10.1613/jair.1.11201

[46] Judea Pearl. 1985. Bayesian Networks: A Model of Self-Activated Memory for Evidential Reasoning. In *Conference of the Cognitive Science Society*.

[47] Stuart Pernsteiner, Calvin Loncaric, Emina Torlak, Zachary Tatlock, Xi Wang, Michael D. Ernst, and Jonathan Jacky. 2016. Investigating Safety of a Radiotherapy Machine Using System Models with Pluggable Checkers. In *Computer*

Aided Verification (CAV). https://doi.org/10.1007/978-3-319-41540-6_2

[48] Phitchaya Mangpo Phothilimthana, Aditya Thakur, Rastislav Bodík, and Dinakar Dhurjati. 2016. Scaling up Superoptimization. In *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. https://doi.org/10.1145/2872362.2872387

[49] Andrew M Pitts. 2013. *Nominal Sets: Names and Symmetry in Computer Science*. Cambridge University Press.

[50] Sorawee Porncharoenwase, James Bornholt, and Emina Torlak. 2020. Fixing Code That Explodes Under Symbolic Evaluation. In *Verification, Model Checking, and Abstract Interpretation (VMCAI)*. https://doi.org/10.1007/978-3-030-39322-9_3

[51] Sorawee Porncharoenwase, Luke Nelson, Xi Wang, and Emina Torlak. 2022. A Formal Foundation for Symbolic Evaluation with Merging. In *Principles of Programming Languages (POPL)*. https://doi.org/10.1145/3498709

[52] Lawrence R Rabiner. 1989. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proc. IEEE*. https://doi.org/10.1109/5.18626

[53] Dan Roth. 1996. On the Hardness of Approximate Reasoning. *Artificial Intelligence*. https://doi.org/10.1016/0004-3702(94)00092-1

[54] Colin Runciman. 1989. What About the Natural Numbers? *Computer Languages*. https://doi.org/10.1016/0096-0551(89)90004-0

[55] Feras A Saad, Martin C Rinard, and Vikash K Mansinghka. 2021. SPPL: Probabilistic Programming with Fast Exact Symbolic Inference. In *Programming Language Design and Implementation (PLDI)*. https://doi.org/10.1145/3453483.3454078

[56] Tian Sang, Paul Beame, and Henry A Kautz. 2005. Performing Bayesian Inference by Weighted Model Counting. In *AAAI Conference on Artificial Intelligence (AAAI)*.

[57] Steffen Smolka, Praveen Kumar, David M. Kahn, Nate Foster, Justin Hsu, Dexter Kozen, and Alexandra Silva. 2019. Scalable Verification of Probabilistic Networks. In *Programming Language Design and Implementation (PLDI)*. https://doi.org/10.1145/3314221.3314639

[58] Fabio Somenzi. 1998. CUDD: CU Decision Diagram Package Release. https://add-lib.scce.info/assets/documents/cudd-manual.pdf

[59] Zachary Susag, Sumit Lahiri, Justin Hsu, and Subhajit Roy. 2022. Symbolic Execution for Randomized Programs. In *Object-Oriented Programming, Systems, Languages and Applications (OOPSLA)*. https://doi.org/10.1145/3563344

[60] Emina Torlak and Rastislav Bodik. 2013. Growing Solver-Aided Languages with Rosette. In *Onward!* https://doi.org/10.1145/2509578.2509586

[61] Emina Torlak and Rastislav Bodík. 2014. A Lightweight Symbolic Virtual Machine for Solver-Aided Host Languages. In *Programming Language Design and Implementation (PLDI)*. https://doi.org/10.1145/2594291.2594340

[62] Erik Voogd, Einar Broch Johnsen, Alexandra Silva, Zachary J Susag, and Andrzej Wąsowski. 2023. Symbolic Semantics for Probabilistic Programs. In *International Conference on Quantitative Evaluation of Systems*. https://doi.org/10.1007/978-3-031-43835-6_23

[63] Mitchell Wand, Ryan Culpepper, Theophilos Giannakopoulos, and Andrew Cobb. 2018. Contextual Equivalence for a Probabilistic Language with Continuous Random Variables and Recursion. In *International Conference on Functional Programming (ICFP)*. https://doi.org/10.1145/3236782

[64] Konstantin Weitz, Doug Woos, Emina Torlak, Michael D. Ernst, Arvind Krishnamurthy, and Zachary Tatlock. 2016. Scalable Verification of Border Gateway Protocol Configurations with an SMT Solver. In *Object-Oriented Programming, Systems, Languages and Applications (OOPSLA)*. https://doi.org/10.1145/2983990.2984012

A Syntactic Sugar for Roulette

Roulette alone can be used to perform inference, but there is a gap between the high-level probabilistic programs from Section 2 and the low-level operations, like `define-symbolic` and `infer`, shown in Section 3. This section closes that gap by constructing a thin layer of syntactic sugar for Roulette, fully realizing it as a high-level PPL. Racket's advanced macro system more than suffices to build this layer.

A.1 Flip

The most basic operation for a discrete PPL is `flip`, which can be written as a simple function:

```
(define (flip pr)
  (for/all ([pr* pr])
    (define-symbolic* x (bern pr*))
    x))
```

Given a probability, `flip` returns fresh symbolic values associated with a Bernoulli distribution parameterized by the input probability. Two aspects of `flip` deserve mention. First, `flip` uses `define-symbolic*` instead of `define-symbolic`. With `define-symbolic*`, a new symbolic value is created each call. With `define-symbolic`, a new symbolic value is created for each lexical occurrence of the form. Second, `flip` uses Rosette's `for/all` construct to support symbolic input probabilities. The distribution given to `bern` must be a concrete value, not a symbolic union. Therefore, `flip` unpacks the symbolic union using `for/all` which maps over a symbolic union by binding each concrete value in the union to `pr*` and evaluating the body, placing the result in a new symbolic union under the same guards.

Consider the following program, a *hierarchical model*:

```
(define p (if (flip 1/2) 1/4 3/4)) (if (flip p) 'a 'b)
```

Here, `flip` generates two new symbolic values (x_1 and x_2) and produces this symbolic union:

$$\left[(x_0 \wedge x_1) \vee (\neg x_0 \wedge x_2) : \texttt{'a}, \quad (x_0 \wedge \neg x_1) \vee (\neg x_0 \wedge \neg x_2) : \texttt{'b}\right].$$

A.2 Observation

From a programming-language perspective, `observe!` is like `assert`.[3] In a language like C, assertions raise an error when given `false`. In a PPL, `observe!` prunes all branches of computation where the given random variable is `false`. Consider the following implementation of observation and delimited observation:

```
(define evidence #true)

(define (observe! v)
  (set! evidence (and evidence v)))

(define (query e)
  (define unnormalized (infer (if evidence e '⊥)))
  (define normalizer (- 1 (unnormalized '⊥)))
  (for/pmf ([(value weight) (in-pmf unnormalized)]
            #:unless (eq? value '⊥))
    (values value (/ weight normalizer))))
```

[3]Indeed, Rosette's built-in `assert` form can be used to implement observation. The code in this section does not do so since it makes delimited observation trickier.

A global mutable variable named `evidence` stores the random variable used for conditioning. Applying `observe!` to a random variable mutably conjoins it to the existing `evidence`. When querying the probability of a random variable, the given value is conditioned on the evidence using an `if`. Probability mass that does not satisfy the evidence is sent to `'⊥`.[4] After computing the `result`, the `query` function renormalizes the PMF with a `for/pmf` form that divides probabilities by a normalizing constant. Because Roulette soundly handles mutation, unusual cases such as observation under a conditional work correctly by default.

Using macros, it is easy to implement advanced language features. For example, delimited observation, where observed evidence is contained within a particular dynamic extent, can implemented as such:

```
(define-syntax-rule (with-observe body ...+)
  (let ([old evidence])
    (begin0
      (begin body ...+)
      (set! evidence old))))
```

Before the `body` expressions are evaluated, `evidence` is saved to a local variable. During evaluation of the `body` expressions, observations are recorded in `evidence`. Afterwards, `evidence` is reset to its original value and the value of the last `body` expression is returned using a combination of `begin` and `begin0`.[5]

A.3 Expectation

Expectation uses the result of `query` to compute the expected value of a random variable:

```
(define (expectation v)
  (for/sum ([(val prob) (in-pmf (query v))])
    (* val prob)))
```

Using Racket's `for/sum` form makes it simple to compute a weighted sum. Because `expectation` uses `query` and not `infer`, the expected value works in the presence of observation and delimited observation.

[4] In the actual implementation, this is a not the symbol `'⊥` but a symbol generated using `gensym`.
[5] A robust implementation of delimited forms in Scheme or Racket would use `dynamic-wind` to properly handle control effects. Rosette does not support `dynamic-wind`, but this alternative is good enough for practical purposes.

$\boxed{\rho, s \vdash (e, \sigma) \Downarrow (v, \sigma')}$

$$\textsc{Var}\ \frac{}{\rho, s \vdash (x, \sigma) \Downarrow (\rho(x), \sigma)} \qquad \textsc{Let}\ \frac{s \rightsquigarrow s_1, s_2 \quad \rho, s_1 \vdash (e_1, \sigma) \Downarrow (v_1, \sigma_1) \quad \rho[x \mapsto v_1], s_2 \vdash (e_2, \sigma_1) \Downarrow (v_2, \sigma_2)}{\rho, s \vdash (\texttt{let}\ x = e_1\ \texttt{in}\ e_2, \sigma) \Downarrow (v_2, \sigma_2)}$$

$$\textsc{Lam}\ \frac{}{\rho, s \vdash (\lambda x.e, \sigma) \Downarrow (\texttt{clo}(\lambda x.e, \rho), \sigma)} \qquad \textsc{App}\ \frac{\rho(x) = \texttt{clo}(\lambda x'.\, e', \rho') \quad \rho'[x' \mapsto \rho(y)], s \vdash (e', \sigma) \Downarrow (v, \sigma')}{\rho, s \vdash (x\ y, \sigma) \Downarrow (v, \sigma')}$$

$$\textsc{True}\ \frac{}{\rho, s \vdash (\texttt{true}, \sigma) \Downarrow (\texttt{true}, \sigma)} \qquad \textsc{False}\ \frac{}{\rho, s \vdash (\texttt{false}, \sigma) \Downarrow (\texttt{false}, \sigma)} \qquad \textsc{IfTrue}\ \frac{\rho(x) = \texttt{true} \quad \rho, s \vdash (e_1, \sigma) \Downarrow (v, \sigma')}{\rho, s \vdash (\texttt{if}\ x\ e_1\ e_2, \sigma) \Downarrow (v, \sigma')}$$

$$\textsc{IfFalse}\ \frac{\rho(x) = \texttt{false} \quad \rho, s \vdash (e_2, \sigma) \Downarrow (v, \sigma')}{\rho, s \vdash (\texttt{if}\ x\ e_1\ e_2, \sigma) \Downarrow (v, \sigma')} \qquad \textsc{Num}\ \frac{r \in \mathbb{Q}}{\rho, s \vdash (r, \sigma) \Downarrow (r, \sigma)} \qquad \textsc{Arith}\ \frac{\rho(x), \rho(y) \in \mathbb{Q} \quad \oplus \in \{+, -, \times\}}{\rho, s \vdash (x \oplus y, \sigma) \Downarrow (\rho(x) \llbracket \oplus \rrbracket \rho(y), \sigma)}$$

$$\textsc{Pair}\ \frac{}{\rho, s \vdash ((x, y), \sigma) \Downarrow ((\rho(x), \rho(y)), \sigma)} \qquad \textsc{Fst}\ \frac{\rho(x) = (v, w)}{\rho, s \vdash (\texttt{fst}\, x, \sigma) \Downarrow (v, \sigma)} \qquad \textsc{Snd}\ \frac{\rho(x) = (v, w)}{\rho, s \vdash (\texttt{snd}\, x, \sigma) \Downarrow (w, \sigma)}$$

$$\textsc{Ref}\ \frac{\ell \notin \text{locs}(\rho, \sigma)}{\rho, s \vdash (\texttt{ref}\, x, \sigma) \Downarrow (\ell, \sigma[\ell \mapsto \rho(x)])} \qquad \textsc{Get}\ \frac{\rho(x) \in \text{dom}(\sigma)}{\rho, s \vdash (!x, \sigma) \Downarrow (\sigma(\rho(x)), \sigma)}$$

$$\textsc{Set}\ \frac{\rho(x) \in \text{dom}(\sigma)}{\rho, s \vdash (x := y, \sigma) \Downarrow ((), \sigma[\rho(x) \mapsto \rho(y)])} \qquad \textsc{SymTrue}\ \frac{}{\rho, \mathsf{T} :: s \vdash (\texttt{sym}, \sigma) \Downarrow (\texttt{true}, \sigma)} \qquad \textsc{SymFalse}\ \frac{}{\rho, \mathsf{F} :: s \vdash (\texttt{sym}, \sigma) \Downarrow (\texttt{false}, \sigma)}$$

Fig. 15. Concrete semantics of idealized Rosette.

B Idealized Rosette

B.1 Syntax

$$\begin{aligned}
e \in \text{Expr} ::= {} & x \mid \texttt{let}\ x = e\ \texttt{in}\ e \mid \lambda x.e \mid x\ y \\
& \mid \texttt{true} \mid \texttt{false} \mid \texttt{if}\ x\ e\ e \\
& \mid r \mid x + y \mid x - y \mid x \times y \\
& \mid () \mid (x, y) \mid \texttt{fst}\, x \mid \texttt{snd}\, x \\
& \mid \texttt{ref}\, x \mid\ !y \mid x := y \\
& \mid \texttt{sym} \mid \texttt{fail} \\
\rho \in \text{Env} = {} & \text{Var} \to_{\text{fin}} \text{Val} \\
v \in \text{Val} ::= {} & \texttt{true} \mid \texttt{false} \mid \texttt{clo}(\lambda x.e, \rho) \mid r \mid () \mid (v, v) \mid \ell \in \text{Loc} \\
\sigma \in \text{Store} = {} & \text{Loc} \to_{\text{fin}} \text{Val} \\
r \in {} & \mathbb{Q} \\
s \in \text{BitStream} = {} & [0, 1]^{\mathbb{N}}
\end{aligned}$$

B.2 Concrete semantics

Figure 15 contains the evaluation rules. Notably, there is no rule for `fail`.

B.3 Abstract semantics

Figure 16 contains the evaluation rules.

Definition B.1. Given an infinite supply of symbolic variables, $\alpha, \beta, \ldots \in \mathsf{SymVar}$, let $\mathsf{Model} = \mathsf{SymVar} \to \mathsf{Bool}$.

Definition B.2. For S a subset of SymVar, let Model_S be the set $S \to \mathsf{Bool}$.

Definition B.3. For a set A, the set of *symbolic values of type* A, written $\widehat{A}$, is the set of functions $\mathsf{Model} \to A_\perp$ with finite image and finite support.

Definition B.4. A *symbolic proposition* is an element of $\widehat{\mathsf{Bool}}$. Two symbolic propositions φ, ψ are *disjoint* if $\neg(\varphi(m) \wedge \psi(m))$ for all $m \in \mathsf{Model}$.

Definition B.5. For a finite family $(\varphi_i)_{i \in I}$ of pairwise-disjoint symbolic propositions, and a family of symbolic values $(\widehat{a}_i)_{i \in I}$, their *symbolic union* $[\varphi_i : \widehat{a}_i]_{i \in I}$ is the function defined by:

$$[\varphi_i : \widehat{a}_i]_{i \in I}(m) = \begin{cases} \widehat{a}_i(m), & \varphi_i(m) \text{ for some } i \\ \perp, & \text{otherwise} \end{cases}$$

Lemma B.6 (Determinism)**.** If $\widehat{\rho} \vdash (e, \widehat{\sigma}) \Downarrow (\widehat{v}_1, \widehat{\sigma}'_1, \psi_1)$ and $\widehat{\rho} \vdash (e, \widehat{\sigma}) \Downarrow (\widehat{v}_2, \widehat{\sigma}'_2, \psi_2)$ then $(\widehat{v}_1, \widehat{\sigma}'_1, \psi_1) = (\widehat{v}_2, \widehat{\sigma}'_2, \psi_2)$.

Proof. By induction on $\widehat{\rho} \vdash (e, \widehat{\sigma}) \Downarrow (\widehat{v}_1, \widehat{\sigma}'_1, \psi_1)$ and inversion on $\widehat{\rho} \vdash (e, \widehat{\sigma}) \Downarrow (\widehat{v}_2, \widehat{\sigma}'_2, \psi_2)$. □

B.4 Correctness

Definition B.7. Let $\mathsf{Aut}_{\mathsf{fin}}(\mathsf{Loc})$ be the group of finite permutations of locations — that is, bijective functions $\pi : \mathsf{Loc} \to \mathsf{Loc}$ with $\{\ell \mid \pi(\ell) \neq \ell\}$ finite — under function composition.

Definition B.8. A *nominal set* [49] is a set X with a left action $(\cdot) : \mathsf{Aut}_{\mathsf{fin}}(\mathsf{Loc}) \times X \to X$ by $\mathsf{Aut}_{\mathsf{fin}}(\mathsf{Loc})$ such that for every $x \in X$ there exists a finite set of locations L, called *a support for* x, such that $\pi \cdot x = x$ for every $\pi \in \mathsf{Aut}_{\mathsf{fin}}(\mathsf{Loc})$ with $\{\ell \mid \pi(\ell) = \ell\} \subseteq L$.

Definition B.9. Given a nominal set X, let $\langle \mathsf{Locs} \rangle X$ be the set $\{(L, x) \mid x \in X, L \subseteq \mathsf{locs}(x)\}$ quotiented by the equivalence relation $(\{\ell_1, \ldots, \ell_n\}, x) \sim (\{\ell'_1, \ldots, \ell'_n\}, x')$ iff there exist locations $f_1, \ldots, f_n$ disjoint from $\mathsf{locs}(x, x')$ such that $(\ell_1\ f_1) \ldots (\ell_n\ f_n) \cdot x = (\ell'_1\ f_1) \ldots (\ell'_n\ f_n) \cdot x'$. Elements of the quotient will be written $\langle L \rangle x$. This forms a nominal set, with action $\pi \cdot \langle L \rangle x = \langle \pi(L) \rangle (\pi \cdot x)$.

Proof. This construction mirrors Pitts [49, Definition 4.2], generalized from binding a single location to binding a set of locations. The relation $(\sim)$ is equivariant because swapping is equivariant, so that if $(\ell_1\ f_1) \ldots (\ell_n\ f_n) \cdot x = (\ell'_1\ f_1) \ldots (\ell'_n\ f_n) \cdot x'$ witnesses $(\{\ell_1, \ldots, \ell_n\}, x) \sim (\{\ell'_1, \ldots, \ell'_n\}, x')$ then $(\pi(\ell_1)\ \pi(f_1)) \ldots (\pi(\ell_n)\ \pi(f_n)) \cdot (\pi \cdot x) = (\pi(\ell'_1)\ \pi(f_1)) \ldots (\pi(\ell'_n)\ \pi(f_n)) \cdot (\pi \cdot x')$ witnesses $(\{\pi(\ell_1), \ldots, \pi(\ell_n)\}, \pi \cdot x) \sim (\{\pi(\ell'_1), \ldots, \pi(\ell'_n)\}, \pi(x'))$. The relation $(\sim)$ is an equivalence: reflexivity and symmetry are straightforward; for transitivity,

$\boxed{\widehat{\rho} \vdash (e, \widehat{\sigma}) \Downarrow (\widehat{v}, \widehat{\sigma}', \psi)}$

$$\textsc{Var}\ \frac{}{\widehat{\rho} \vdash (x, \widehat{\sigma}) \Downarrow (\widehat{\rho}(x), \widehat{\sigma}, \mathsf{T})} \qquad \textsc{Let}\ \frac{\widehat{\rho} \vdash (e_1, \widehat{\sigma}) \Downarrow (\widehat{v}_1, \widehat{\sigma}_1, \psi_1) \quad \widehat{\rho}[x \mapsto \widehat{v}_1] \vdash (e_2, \widehat{\sigma}_1) \Downarrow (\widehat{v}_2, \widehat{\sigma}_2, \psi_2)}{\widehat{\rho} \vdash (\mathtt{let}\ x = e_1\ \mathtt{in}\ e_2, \widehat{\sigma}) \Downarrow ([\psi_1 : \widehat{v}_2], [\psi_1 : \widehat{\sigma}_2], \psi_1 \wedge \psi_2)}$$

$$\textsc{Lam}\ \frac{}{\widehat{\rho} \vdash (\lambda x.e, \widehat{\sigma}) \Downarrow ([\mathsf{T} : \mathtt{clo}(\lambda x.e, \widehat{\rho})], \widehat{\sigma}, \mathsf{T})} \qquad \textsc{App}\ \frac{\widehat{\rho}(x) = [\varphi_i : \mathtt{clo}(\lambda x_i.e_i, \widehat{\rho}_i)]_{i \in I} \uplus_{\mathsf{Closure}} \widehat{v} \quad \forall i \in I.\ \widehat{\rho}_i[x_i \mapsto \widehat{\rho}(y)] \vdash (e_i, \widehat{\sigma}) \Downarrow (\widehat{v}_i, \widehat{\sigma}_i, \psi_i)}{\widehat{\rho} \vdash (x\ y, \widehat{\sigma}) \Downarrow ([\varphi_i : \widehat{v}_i]_{i \in I}, [\varphi_i : \widehat{\sigma}_i]_{i \in I}, \bigvee_i (\varphi_i \wedge \psi_i))}$$

$$\textsc{True}\ \frac{}{\widehat{\rho} \vdash (\mathtt{true}, \widehat{\sigma}) \Downarrow ([\mathsf{T} : \mathtt{true}], \widehat{\sigma}, \mathsf{T})} \qquad \textsc{False}\ \frac{}{\widehat{\rho} \vdash (\mathtt{false}, \widehat{\sigma}) \Downarrow ([\mathsf{T} : \mathtt{false}], \widehat{\sigma}, \mathsf{T})}$$

$$\textsc{If}\ \frac{\widehat{\rho}(x) = [\varphi_1 : \mathtt{true}, \varphi_2 : \mathtt{false}] \uplus_{\mathsf{Bool}} \widehat{v} \quad \widehat{\rho} \vdash (e_1, \widehat{\sigma}) \Downarrow (\widehat{v}_1, \widehat{\sigma}_1, \psi_1) \quad \widehat{\rho} \vdash (e_2, \widehat{\sigma}) \Downarrow (\widehat{v}_2, \widehat{\sigma}_2, \psi_2)}{\widehat{\rho} \vdash (\mathtt{if}\ x\ e_1\ e_2, \widehat{\sigma}) \Downarrow ([\varphi_1 : \widehat{v}_1, \varphi_2 : \widehat{v}_2], [\varphi_1 : \widehat{\sigma}_1, \varphi_2 : \widehat{\sigma}_2], (\varphi_1 \wedge \psi_1) \vee (\varphi_2 \wedge \psi_2))}$$

$$\textsc{Num}\ \frac{r \in \mathbb{Q}}{\widehat{\rho} \vdash (r, \widehat{\sigma}) \Downarrow ([\mathsf{T} : r], \widehat{\sigma}, \mathsf{T})}$$

$$\textsc{Arith}\ \frac{\widehat{\rho}(x) = [\varphi_i : r_i]_{i \in I} \uplus_{\mathsf{Num}} \widehat{v}_1 \quad \widehat{\rho}(y) = [\varphi_j : r_j]_{j \in J} \uplus_{\mathsf{Num}} \widehat{v}_2 \quad \forall\, i \in I, j \in J.\ r_i, r_j \in \mathbb{Q} \quad \oplus \in \{+, -, \times\}}{\widehat{\rho} \vdash (x \oplus y, \widehat{\sigma}) \Downarrow ([\varphi_i \wedge \varphi_j : r_i \llbracket \oplus \rrbracket\, r_j]_{i \in I, j \in J}, \widehat{\sigma}, \bigvee_{i,j} (\varphi_i \wedge \varphi_j))}$$

$$\textsc{Pair}\ \frac{}{\widehat{\rho} \vdash ((x, y), \widehat{\sigma}) \Downarrow ((\widehat{\rho}(x), \widehat{\rho}(y)), \widehat{\sigma}, \mathsf{T})}$$

$$\textsc{Fst}\ \frac{\widehat{\rho}(x) = [\varphi_i : (v_i, w_i)]_{i \in I} \uplus_{\mathsf{Pair}} \widehat{v}}{\widehat{\rho} \vdash (\mathtt{fst}\, x, \widehat{\sigma}) \Downarrow ([\varphi_i : v_i]_{i \in I}, \widehat{\sigma}, \bigvee_{i \in I} \varphi_i)} \qquad \textsc{Snd}\ \frac{\widehat{\rho}(x) = [\varphi_i : (v_i, w_i)]_{i \in I} \uplus_{\mathsf{Pair}} \widehat{v}}{\widehat{\rho} \vdash (\mathtt{snd}\, x, \widehat{\sigma}) \Downarrow ([\varphi_i : w_i]_{i \in I}, \widehat{\sigma}, \bigvee_{i \in I} \varphi_i)}$$

$$\textsc{Ref}\ \frac{\ell \text{ smallest not in } \mathrm{locs}(\widehat{\rho}, \widehat{\sigma})}{\widehat{\rho} \vdash (\mathtt{ref}\, x, \widehat{\sigma}) \Downarrow ([\mathsf{T} : \ell], \widehat{\sigma}[\ell \mapsto \widehat{\rho}(x)], \mathsf{T})} \qquad \textsc{Get}\ \frac{\widehat{\rho}(x) = [\varphi_i : \ell_i]_{i \in I} \uplus_{\mathsf{Loc}} \widehat{v}}{\widehat{\rho} \vdash (!x, \widehat{\sigma}) \Downarrow ([\varphi_i : \widehat{\sigma}(\ell_i)]_{i \in I}, \widehat{\sigma}, \bigvee_i \varphi_i)}$$

$$\textsc{Set}\ \frac{\widehat{\rho}(x) = [\varphi_i : \ell_i]_{i \in I} \uplus_{\mathsf{Loc}} \widehat{v}}{\widehat{\rho} \vdash (x := y, \widehat{\sigma}) \Downarrow ([(\bigvee_i \varphi_i) : ()], [\varphi_i : \widehat{\sigma}[\ell_i \mapsto \widehat{\rho}(y)]]_{i \in I}, \bigvee_i \varphi_i)} \qquad \textsc{Sym}\ \frac{\alpha \text{ smallest not in } \mathrm{symvars}(\widehat{\rho}, \widehat{\sigma})}{\widehat{\rho} \vdash (\mathtt{sym}, \widehat{\sigma}) \Downarrow ([\alpha : \mathtt{true}, \neg\alpha : \mathtt{false}], \widehat{\sigma}, \mathsf{T})}$$

$$\textsc{Fail}\ \frac{}{\widehat{\rho} \vdash (\mathtt{fail}, \widehat{\sigma}) \Downarrow (\varnothing, \varnothing, \mathsf{F})}$$

Fig. 16. Abstract semantics of idealized Rosette.

$$\begin{aligned}
&(\{\ell_1, \ldots, \ell_n\}, x) \sim (\{\ell_1', \ldots, \ell_n'\}, x') \wedge (\{\ell_1', \ldots, \ell_n'\}, x) \sim (\{\ell_1'', \ldots, \ell_n''\}, x') \\
&\iff \begin{pmatrix} (\mathcal{H} f_1 \ldots \mathcal{H} f_n.\ (\ell_1\ f_1) \ldots (\ell_n\ f_n) \cdot x = (\ell_1'\ f_1) \ldots (\ell_n'\ f_n) \cdot x') \\ \wedge\ (\mathcal{H} f_1 \ldots \mathcal{H} f_n.\ (\ell_1'\ f_1) \ldots (\ell_n'\ f_n) \cdot x' = (\ell_1''\ f_1) \ldots (\ell_n''\ f_n) \cdot x'') \end{pmatrix} \\
&\overset{\text{Pitts [49, Proposition 3.10]}}{\iff} \begin{pmatrix} \mathcal{H} f_1 \ldots \mathcal{H} f_n. \\ (\ell_1\ f_1) \ldots (\ell_n\ f_n) \cdot x = (\ell_1'\ f_1) \ldots (\ell_n'\ f_n) \cdot x' \wedge \\ (\ell_1'\ f_1) \ldots (\ell_n'\ f_n) \cdot x' = (\ell_1''\ f_1) \ldots (\ell_n''\ f_n) \cdot x'' \end{pmatrix} \\
&\implies \mathcal{H} f_1 \ldots \mathcal{H} f_n . (\ell_1\ f_1) \ldots (\ell_n\ f_n) \cdot x = (\ell_1''\ f_1) \ldots (\ell_n''\ f_n) \cdot x'' \\
&\iff (\{\ell_1, \ldots, \ell_n\}, x) \sim (\{\ell_1'', \ldots, \ell_n''\}, x').
\end{aligned}$$

This shows $(\sim)$ is an equivariant equivalence relation, so the quotient nominal set is well-defined, with action as claimed. □

Definition B.10. The sets Env and Val form nominal sets via the following group action:

$$\begin{aligned}
\pi \cdot \rho &= \{x \mapsto \pi \cdot \rho(x) \mid x \in \mathsf{dom}(\rho)\} \\
\pi \cdot \mathtt{true} &= \mathtt{true} \\
\pi \cdot \mathtt{false} &= \mathtt{false} \\
\pi \cdot \mathtt{clo}(\lambda x.e, \rho') &= \mathtt{clo}(\lambda x.e, \pi \cdot \rho') \\
\pi \cdot r &= r && r \in \mathbb{Q} \\
\pi \cdot () &= () \\
\pi \cdot (v, w) &= (\pi \cdot v, \pi \cdot w) \\
\pi \cdot \ell &= \pi(\ell)
\end{aligned}$$

Definition B.11. The set Store forms a nominal set via the following group action:

$$\pi \cdot \sigma = \{\pi \cdot \ell \mapsto \pi \cdot v \mid \ell \mapsto v \in \sigma\}$$

Definition B.12. Let Result be the nominal set $\langle \mathsf{Locs} \rangle (\mathsf{Val} \times \mathsf{Store})$.

Definition B.13. A tuple $(\rho, e, \sigma) \in \mathsf{Env} \times \mathsf{Exp} \times \mathsf{Store}$ is *well-formed* if $\mathsf{locs}(\rho) \cup \mathsf{locs}(\sigma) \subseteq \mathsf{dom}(\sigma)$, $\mathsf{FV}(e) \subseteq \mathsf{dom}(\rho)$, and $\mathsf{FV}(\rho(x)) = \varnothing$ for all $x \in \mathsf{dom}(\rho)$. Let Config be the nominal set of well-formed tuples.

Proof. The well-formedness condition is equivariant, so this indeed forms a nominal set. □

Lemma B.14 (Invariant of the concrete semantics)**.** If $(\rho, e, \sigma) \in \mathsf{Config}$ and $\rho, s \vdash (e, \sigma) \Downarrow (v, \sigma')$, then $\sigma' = \sigma_{\mathsf{old}} \uplus \sigma_{\mathsf{new}}$ for some $\sigma_{\mathsf{old}}, \sigma_{\mathsf{new}}$ with $\mathsf{dom}(\sigma') = \mathsf{dom}(\sigma_{\mathsf{old}})$.

Proof. By induction on $\rho, s \vdash (e, \sigma) \Downarrow (v, \sigma')$. □

Lemma B.15 (Equivariance)**.** Let $\pi : \mathsf{Loc} \to \mathsf{Loc}$ be a permutation on store locations. Then $\rho, s \vdash (e, \sigma) \Downarrow (v, \sigma')$ implies that $\pi \cdot \rho, s \vdash (e, \pi \cdot \sigma) \Downarrow (\pi \cdot v, \pi \cdot \sigma)$.

Proof. Let $\pi : \mathsf{Loc} \to \mathsf{Loc}$, $\rho \in \mathsf{Env}$, $s \in \mathbb{B}^{\mathbb{N}}$, $e \in \mathsf{Expr}$, $\sigma, \sigma' \in \mathsf{Store}$, and $v \in \mathsf{Value}$ such that

$$\rho, s \vdash (e, \sigma) \Downarrow (v, \sigma')$$

We will show that

$$\pi \cdot \rho, s \vdash (e, \pi \cdot \sigma) \Downarrow (\pi \cdot v, \pi \cdot \sigma')$$

by structural induction on $\rho \vdash (e, \sigma) \Downarrow (v, \sigma')$. We have several cases:

(Var) Consider $e = x$. The only concrete big-step rule that matches $e = x$ is Var. Hence, by inversion of our assumption that $\rho \vdash (e, \sigma) \Downarrow (v, \sigma')$, we get

$$\rho, s \vdash (x, \sigma) \Downarrow (\rho(x), \sigma)$$

where $x \in \mathsf{dom}(\rho)$. Clearly, then, we also have that $x \in \mathsf{dom}(\pi \cdot \rho)$. Hence, we can prove

$$\pi \cdot \rho, s \vdash (x, \pi \cdot \sigma) \Downarrow (\pi \cdot \rho(x), \pi \cdot \sigma)$$

using the concrete big-step judgement Var.

(Lam) Consider $e = \lambda x.e'$. The only concrete rule that matches $e = \lambda x.e'$ is Lam. Hence, by inversion of our assumption that $\rho \vdash (e, \sigma) \Downarrow (v, \sigma')$, we get

$$\rho, s \vdash (\lambda x.e', \sigma) \Downarrow (\texttt{clo}(\lambda x.e', \rho), \sigma)$$

Clearly, then, we can also prove that

$$\pi \cdot \rho, s \vdash (\lambda x.e', \pi \cdot \sigma) \Downarrow (\texttt{clo}(\lambda x.e', \pi \cdot \rho), \pi \cdot \sigma)$$

using the concrete big-step judgement Lam. Note, we have $\pi \cdot \texttt{clo}(\lambda x.e', \rho) = \texttt{clo}(\lambda x.e', \pi \cdot \rho)$ by definition, so we are done.

(True) Consider $e = \texttt{true}$. The only concrete rule that matches $e = \texttt{true}$ is True. Hence, by inversion of our assumption that $\rho \vdash (e, \sigma) \Downarrow (v, \sigma')$, we get

$$\rho, s \vdash (\texttt{true}, \sigma) \Downarrow (\texttt{true}, \sigma)$$

Clearly, then we can also prove that

$$\pi \cdot \rho, s \vdash (\texttt{true}, \pi \cdot \sigma) \Downarrow (\texttt{true}, \pi \cdot \sigma)$$

using the concrete big-step judgement True. Note, we have $\pi \cdot \texttt{true} = \texttt{true}$ by definition, so we are done.

(False) Consider $e = \texttt{false}$. Apply identical reasoning as in the True case.

(Num) Consider $e = a$ for $a \in \mathbb{Q}$. Apply identical reasoning as in the True case.

(Arith) Consider $e = x_1 \oplus x_2$ for $\oplus \in \{+, -, \times\}$. The only concrete rule that matches $e = x_1 \oplus x_2$ is Arith. Hence, by inversion of our assumption that $\rho \vdash (e, \sigma) \Downarrow (v, \sigma')$, we get

$$\rho, s \vdash (x_1 \oplus x_2, \sigma) \Downarrow (\rho(x_1) \llbracket \oplus \rrbracket \rho(x_2), \sigma)$$

where $\rho(x_1), \rho(x_2) \in \mathbb{Q}$. Since $x, y \in \mathsf{dom}(\rho)$, we have that $x, y \in \mathsf{dom}(\pi \cdot \rho)$ by definition. Moreover, because $\rho(x_1), \rho(x_2) \in \mathbb{Q}$, we get that $\pi \cdot \rho(x_1), \pi \cdot \rho(x_2) \in \mathbb{Q}$ by definition. Thus, we can prove that

$$\pi \cdot \rho, s \vdash (x_1 \oplus x_2, \pi \cdot \sigma) \Downarrow (\pi \cdot \rho(x_1) \llbracket \oplus \rrbracket \pi \cdot \rho(x_2), \pi \cdot \sigma)$$

using the concrete big-step judgement Arith. Note,

$$\pi \cdot (\rho(x_1) \llbracket \oplus \rrbracket \rho(x_2)) = \rho(x_1) \llbracket \oplus \rrbracket \rho(x_2) = \pi \cdot \rho(x_1) \llbracket \oplus \rrbracket \pi \cdot \rho(x_2)$$

by definition. Hence, we are done.

(Pair) Consider $e = (x, y)$. The only concrete rule that matches $e = (x, y)$ is Pair. Hence, by inversion of our assumption that $\rho \vdash (e, \sigma) \Downarrow (v, \sigma')$, we get

$$\rho, s \vdash ((x, y), \sigma) \Downarrow ((\rho(x), \rho(y)), \sigma)$$

where $x, y \in \mathsf{dom}(\rho)$. Clearly, because $x, y \in \mathsf{dom}(\rho)$, we also have that $x, y \in \mathsf{dom}(\pi \cdot \rho)$ by definition. Hence, we can prove that

$$\pi \cdot \rho, s \vdash ((x, y), \pi \cdot \sigma) \Downarrow ((\pi \cdot \rho(x), \pi \cdot \rho(y)), \pi \cdot \sigma)$$

using the concrete big-step judgement Pair. Note,

$$\pi \cdot (\rho(x), \rho(y)) = (\pi \cdot \rho(x), \pi \cdot \rho(y))$$

by definition, so we are done.

(Fst) Consider $e = \texttt{fst}\, x$. The only concrete rule that matches $e = \texttt{fst}\, x$ is Fst. Hence, by inversion of our assumption that $\rho \vdash (e, \sigma) \Downarrow (v, \sigma')$, we get

$$\rho, s \vdash (\texttt{fst}\, x, \sigma) \Downarrow (v, \sigma)$$

where $\rho(x) = (v, w)$. Clearly, because $x \in \mathsf{dom}(\rho)$, we have that $x \in \mathsf{dom}(\pi \cdot \rho)$ by definition. Moreover, we know that $\pi \cdot \rho(x) = \pi \cdot (v, w) = (\pi \cdot v, \pi \cdot w)$ by definition. Hence, we can prove that

$$\pi \cdot \rho, s \vdash (\mathtt{fst}\, x, \pi \cdot \sigma) \Downarrow (\pi \cdot v, \pi \cdot \sigma)$$

using the concrete big-step judgement Fst, so we are done.

(Snd) Consider $e = \mathsf{snd}\, x$. Apply identical reasoning as in the Fst case.

(Ref) Consider $e = \mathtt{ref}\, x$. The only concrete rule that matches $e = \mathtt{ref}\, x$ is Ref. Hence, by inversion of our assumption that $\rho \vdash (e, \sigma) \Downarrow (v, \sigma')$, we get

$$\rho, s \vdash (\mathtt{ref}\, x, \sigma) \Downarrow (\ell, \sigma[\ell \mapsto \rho(x)])$$

where $x \in \mathsf{dom}(\rho)$ and ℓ is not in $\mathsf{locs}(\rho, \sigma)$. Clearly, because $x \in \mathsf{dom}(\rho)$, we have that $x \in \mathsf{dom}(\pi \cdot \rho)$ by definition. Now, consider $\ell' = \pi(\ell)$. Notice, because ℓ is not in $\mathsf{locs}(\rho, \sigma)$, it must be the case that ℓ' is not in $\mathsf{locs}(\pi \cdot \rho, \pi \cdot \sigma)$. To see why, suppose, for the purposes of contradiction, that ℓ' is in $\mathsf{locs}(\pi \cdot \rho, \pi \cdot \sigma)$. There are two sub-cases:

(a) If $\ell' \in \mathsf{im}(\pi \cdot \rho)$, then there must exist some mapping $y \mapsto \ell' \in \pi \cdot \rho$. This implies that the mapping $y \mapsto \pi^{-1}(\ell')$, which is equivalently $y \mapsto \ell$, is in ρ, which is a contradiction.

(b) Similarly, if $\ell' \in \mathsf{dom}(\pi \cdot \sigma)$, then there must exist some mapping $\ell' \mapsto v \in \pi \cdot \sigma$. Again, by definition, this means that $\pi^{-1}(\ell') \mapsto \pi^{-1}(v)$, which is equivalently $\ell \mapsto \pi^{-1}(v)$, is in σ, which is a contradiction.

Thus, ℓ' is not in $\mathsf{locs}(\pi \cdot \rho, \pi \cdot \sigma)$. This allows us to prove

$$\pi \cdot \rho, s \vdash (\mathtt{ref}\, x, \pi \cdot \sigma) \Downarrow (\ell', (\pi \cdot \sigma)[\ell' \mapsto \pi \cdot \rho(x)])$$

using the concrete big-step judgement Ref. Note, we have, as previously mentioned, that $\pi \cdot \ell = \pi(\ell) = \ell'$ and that

$$\begin{aligned}
\pi \cdot (\sigma[\ell \mapsto \rho(x)]) &= \pi \cdot (\sigma \uplus \{\ell \mapsto \rho(x)\}) \\
&= (\pi \cdot \sigma) \uplus (\pi \cdot \{\ell \mapsto \rho(x)\}) \\
&= (\pi \cdot \sigma) \uplus \{\ell' \mapsto \pi \cdot \rho(x)\} \\
&= (\pi \cdot \sigma)[\ell' \mapsto \pi \cdot \rho(x)]
\end{aligned}$$

so we are done.

(Get) Consider $e = !x$. The only concrete rule that matches $e = !x$ is Get. Hence, by inversion of our assumption that $\rho \vdash (e, \sigma) \Downarrow (v, \sigma')$, we get

$$\rho, s \vdash (!x, \sigma) \Downarrow (\sigma(\rho(x)), \sigma)$$

where $\rho(x) \in \mathsf{dom}(\sigma)$. As highlighted earlier, $x \in \mathsf{dom}(\rho)$ implies that $x \in \mathsf{dom}(\pi \cdot \rho)$ by definition. Now, say that $\sigma(\rho(x)) = v$ for some $v \in \mathsf{Value}$. Then, if $\rho(x) \in \mathsf{dom}(\sigma)$, we have $\rho(x) \mapsto v \in \sigma$. By definition, we then get $(\pi \cdot \rho(x)) \mapsto (\pi \cdot v) \in \pi \cdot \sigma$, meaning $\pi \cdot \rho(x) \in \mathsf{dom}(\pi \cdot \sigma)$. Hence, we can prove

$$\pi \cdot \rho, s \vdash (!x, \pi \cdot \sigma) \Downarrow ((\pi \cdot \sigma)(\pi \cdot \rho(x)), \pi \cdot \sigma)$$

using the concrete big-step judgement Get, noting that

$$\pi \cdot \sigma(\rho(x)) = \pi \cdot v = (\pi \cdot \sigma)(\pi \cdot \rho(x))$$

by the reasoning above. Thus, we are done.

(Set) Consider $e = x := y$. The only concrete rule that matches $e = x := y$ is Set. Hence, by inversion of our assumption that $\rho \vdash (e, \sigma) \Downarrow (v, \sigma')$, we get

$$\rho, s \vdash (x := y, \sigma) \Downarrow ((), \sigma[\rho(x) \mapsto \rho(y)])$$

where $y \in \mathsf{dom}(\rho)$ and $\rho(x) \in \mathsf{dom}(\sigma)$. Clearly, $y \in \mathsf{dom}(\rho)$ implies that $y \in \mathsf{dom}(\pi \cdot \rho)$. Moreover, as highlighted in the Get case, $\pi \cdot \rho(x) \in \mathsf{dom}(\pi \cdot \sigma)$. Therefore, we can prove

$$\pi \cdot \rho, s \vdash (x \coloneqq y, \pi \cdot \sigma) \Downarrow ((), (\pi \cdot \sigma)[\pi \cdot \rho(x) \mapsto \pi \cdot \rho(y)])$$

using the concrete big-step judgement Set. Note, we have that $\pi \cdot () = ()$ and that

$$\begin{aligned}\pi \cdot \sigma[\rho(x) \mapsto \rho(y)] &= \pi \cdot (\sigma \uplus \{\rho(x) \mapsto \rho(y)\}) \\ &= (\pi \cdot \sigma) \uplus (\pi \cdot \{\rho(x) \mapsto \rho(y)\}) \\ &= (\pi \cdot \sigma) \uplus \{\pi \cdot \rho(x) \mapsto \pi \cdot \rho(y)\} \\ &= (\pi \cdot \sigma)[\pi \cdot \rho(x) \mapsto \pi \cdot \rho(y)]\end{aligned}$$

so we are done.

(SymTrue) Consider $e = \texttt{sym}$ and $b :: s$ with $b = \mathsf{T}$. The only concrete rule that matches $e = \texttt{sym}$ and $b :: s$ with $b = \mathsf{T}$ is SymTrue. Hence, by inversion of our assumption that $\rho \vdash (e, \sigma) \Downarrow (v, \sigma')$, we get

$$\rho, s \vdash (\texttt{sym}, \sigma) \Downarrow (\texttt{true}, \sigma)$$

Because $b = \mathsf{T}$, we can immediately prove that

$$\pi \cdot \rho, b :: s \vdash (\texttt{sym}, \pi \cdot \sigma) \Downarrow (\texttt{true}, \pi \cdot \sigma)$$

using the concrete big-step judgement IfTrue. Note, $\pi \cdot \texttt{true} = \texttt{true}$, so we are done.

(SymFalse) Consider $e = \texttt{sym}$ and $b :: s$ with $b = \mathsf{F}$. Apply identical reasoning as in the SymTrue case.

(IfTrue) Consider $e = \texttt{if}\ x\ e_1\ e_2$ with $\rho(x) = \texttt{true}$. The only concrete rule that matches $e = \texttt{if}\ x\ e_1\ e_2$ with $\rho(x) = \texttt{true}$ is IfTrue. Hence, by inversion of our assumption that $\rho \vdash (e, \sigma) \Downarrow (v, \sigma')$, we get

$$\rho, s \vdash (\texttt{if}\ x\ e_1\ e_2, \sigma) \Downarrow (v, \sigma')$$

with

$$p, s \vdash (e_1, \sigma) \Downarrow (v, \sigma')$$

By the inductive hypothesis, we have that

$$\pi \cdot \rho, s \vdash (e_1, \pi \cdot \sigma) \Downarrow (\pi \cdot v, \pi \cdot \sigma')$$

Additionally, we know that, because $\pi \cdot \rho(x) = \pi \cdot \texttt{true} = \texttt{true}$. Hence, we can prove

$$\pi \cdot \rho, s \vdash (e_1, \pi \cdot \sigma) \Downarrow (\pi \cdot v, \pi \cdot \sigma')$$

using the concrete big-step judgement IfTrue, so we are done.

(IfFalse) Consider $e = \texttt{if}\ x\ e_1\ e_2$ with $\rho(x) = \texttt{false}$. Apply identical reasoning as in the IfFalse case.

(App) Consider $e = x_1 x_2$. The only concrete rule that matches $e = x_1 x_2$ is App. Hence, by inversion of our assumption that $\rho \vdash (e, \sigma) \Downarrow (v, \sigma')$, we get

$$\rho, s \vdash (x_1 x_2, \sigma) \Downarrow (v, \sigma')$$

with $\rho(x_1) = \texttt{clo}(\lambda x'.e', \rho')$ and

$$\rho'[x' \mapsto \rho(x_2)], s \vdash (e', \sigma) \Downarrow (v, \sigma')$$

By the inductive hypothesis, we have that

$$\pi \cdot \rho'[x' \mapsto \rho(x_2)], s \vdash (e', \pi \cdot \sigma) \Downarrow (\pi \cdot v, \pi \cdot \sigma')$$

We are able to rewrite $\pi \cdot \rho'[x' \mapsto \rho(x_2)] = (\pi \cdot \rho')[x' \mapsto \pi \cdot \rho(x_2)]$ by definition. Notice, because $x_1 \in \mathsf{dom}(\rho)$, we know that $x_1 \in \mathsf{dom}(\pi \cdot \rho)$. Moreover, we have that $\pi \cdot \rho(x_1) = \pi \cdot \texttt{clo}(\lambda x'.e', \rho') = \texttt{clo}(\lambda x'.e', \pi \cdot \rho')$. Thus, because

$$\pi \cdot \rho(x_1) = \texttt{clo}(\lambda x'.e', \pi \cdot \rho') \text{ and } (\pi \cdot \rho')[x \mapsto \pi \cdot \rho(x_2)], s \vdash (e', \pi \cdot \sigma) \Downarrow (\pi \cdot v, \pi \cdot \sigma')$$

we can prove that

$$\pi \cdot \rho, s \vdash (x_1 x_2, \pi \cdot \sigma) \Downarrow (\pi \cdot v, \pi \cdot \sigma')$$

Thus, we are done.

(Let) Let $e = \texttt{let}\ x = e_1\ \texttt{in}\ e_2$. The only concrete rule that matches $e = \texttt{let}\ x = e_1\ \texttt{in}\ e_2$ is Let. Hence, by inversion of our assumption that $\rho \vdash (e, \sigma) \Downarrow (v, \sigma')$, we get

$$\rho, s \vdash (\texttt{let}\ x = e_1\ \texttt{in}\ e_2, \sigma) \Downarrow (v_2, \sigma_2)$$

where, for $s \rightsquigarrow s_1, s_2$, we have

$$\rho, s_1 \vdash (e_1, \sigma) \Downarrow (v_1, \sigma_1)$$

and

$$\rho[x \mapsto v_1], s_2 \vdash (e_2, \sigma_1) \Downarrow (v_2, \sigma_2)$$

By the inductive hypothesis, we have that

$$\pi \cdot \rho, s_1 \vdash (e_1, \pi \cdot \sigma) \Downarrow (\pi \cdot v_1, \pi \cdot \sigma_1)$$

and

$$\pi \cdot \rho[x \mapsto v_1], s_2 \vdash (e_2, \pi \cdot \sigma_1) \Downarrow (\pi \cdot v_2, \pi \cdot \sigma_2)$$

We can rewrite

$$\begin{aligned} \pi \cdot \rho[x \mapsto v_1] &= \pi \cdot \left(\rho \uplus \{x \mapsto v_1\}\right) \\ &= (\pi \cdot \rho) \uplus (\pi \cdot \{x \mapsto v_1\}) \\ &= (\pi \cdot \rho) \uplus \{x \mapsto \pi \cdot v_1\} \\ &= (\pi \cdot \rho)[x \mapsto \pi \cdot v_1] \end{aligned}$$

Hence, because $s \rightsquigarrow s_1, s_2$ and

$$\pi \cdot \rho, s_1 \vdash (e_1, \pi \cdot \sigma) \Downarrow (\pi \cdot v_1, \pi \cdot \sigma_1) \text{ and } (\pi \cdot \rho)[x \mapsto \pi \cdot v_1] \vdash (e_2, \pi \cdot \sigma_1) \Downarrow (\pi \cdot v_2, \pi \cdot \sigma_2)$$

we can prove that

$$\pi \cdot \rho, s \vdash (\texttt{let}\ x = e_1\ \texttt{in}\ e_2, \pi \cdot \sigma) \Downarrow (\pi \cdot v_2, \pi \cdot \sigma_2)$$

Hence, we are done.

This completes our proof by induction. □

Corollary B.16. The function $\mathsf{run} : \mathsf{Config} \to \mathsf{Result}_\perp$ is a map of nominal sets.

Proof. It suffices to show $\pi \cdot \mathsf{run}(\rho, e, \sigma)(s) = \mathsf{run}(\pi \cdot \rho, e, \pi \cdot \sigma)(s)$ for all $\pi : \mathsf{Aut}_{\mathsf{fin}}(\mathsf{Loc})$. Let $\rho \in \mathsf{Env}$, $e \in \mathsf{Expr}$, $\sigma \in \mathsf{Store}$, $s \in \mathbb{B}^{\mathbb{N}}$, and $\pi : \mathsf{Loc} \to \mathsf{Loc}$ such that $\pi(\ell) \neq \ell$ for a finite number of elements $\ell \in \mathsf{Loc}$. We have two cases:

(1) Suppose that $\mathsf{run}(\rho, e, \sigma)(s) = \perp$. Then it suffices to show that $\mathsf{run}(\pi \cdot \rho, e, \pi \cdot \sigma)(s) = \perp$. Assume, for the purposes of contradiction, that

$$\mathsf{run}(\pi \cdot \rho, e, \pi \cdot \sigma)(s) = \langle \mathsf{dom}(\sigma') \setminus \mathsf{dom}(\pi \cdot \sigma) \rangle (v, \sigma')$$

This implies, by the definition of run, that

$$\pi \cdot \rho, s \vdash (e, \pi \cdot \sigma) \Downarrow (v, \sigma')$$

Now, consider $\pi^{-1} : \mathsf{Loc} \to \mathsf{Loc}$. By Theorem B.15, we have that

$$\rho, s \vdash (e, \sigma) \Downarrow (\pi^{-1} \cdot v, \pi^{-1} \cdot \sigma')$$

since $\pi^{-1} \cdot \pi \cdot \rho = \rho$ and $\pi^{-1} \cdot \pi \cdot \sigma = \sigma$. Thus, by definition of run, we have that

$$\mathsf{run}(\rho, e, \sigma)(s) = \langle \mathsf{dom}(\pi^{-1} \cdot \sigma') \setminus \mathsf{dom}(\sigma) \rangle (\pi^{-1} \cdot v, \pi^{-1} \cdot \sigma')$$

This is a contradiction, since we assumed $\mathsf{run}(\rho, e, \sigma)(s) = \bot$. Therefore, our assumption that $\mathsf{run}(\pi \cdot \rho, e, \pi \cdot \sigma)(s) = \langle \mathsf{dom}(\sigma') \setminus \mathsf{dom}(\pi \cdot \sigma)\rangle(v, \sigma')$ was wrong, and we conclude $\mathsf{run}(\pi \cdot \rho, e, \pi \cdot \sigma)(s) = \bot$.

This shows $\pi \cdot \mathsf{run}(\rho, e, \sigma)(s) = \pi \cdot \bot = \bot = \mathsf{run}(\pi \cdot \rho, e, \pi \cdot \sigma)(s)$, as desired.

(2) Suppose that $\mathsf{run}(\rho, e, \sigma) = \langle \mathsf{dom}(\sigma') \setminus \mathsf{dom}(\sigma)\rangle(v, \sigma')$. Then, by the definition of run, we have

$$\rho, s \vdash (e, \sigma) \Downarrow (v, \sigma')$$

By Theorem B.15, we then get

$$\pi \cdot \rho, s \vdash (e, \pi \cdot \sigma)(s) \Downarrow (\pi \cdot v, \pi \cdot \sigma')$$

This implies, by the definition of run, that

$$\mathsf{run}(\pi \cdot \rho, e, \pi \cdot \sigma)(s) = \langle \mathsf{dom}(\pi \cdot \sigma') \setminus \mathsf{dom}(\pi \cdot \sigma)\rangle(\pi \cdot v, \pi \cdot \sigma')$$

It now suffices to show that

$$\pi \cdot \langle \mathsf{dom}(\sigma') \setminus \mathsf{dom}(\sigma)\rangle(v, \sigma') = \langle \mathsf{dom}(\pi \cdot \sigma') \setminus \mathsf{dom}(\pi \cdot \sigma)\rangle(\pi \cdot v, \pi \cdot \sigma')$$

But we also have, by Theorem B.9, that

$$\pi \cdot \langle \mathsf{dom}(\sigma') \setminus \mathsf{dom}(\sigma)\rangle(v, \sigma') = \langle \pi \cdot (\mathsf{dom}(\sigma') \setminus \mathsf{dom}(\sigma))\rangle(\pi \cdot v, \pi \cdot \sigma')$$

So it is enough to show that $\pi \cdot (\mathsf{dom}(\sigma') \setminus \mathsf{dom}(\sigma)) = \mathsf{dom}(\pi \cdot \sigma') \setminus \mathsf{dom}(\pi \cdot \sigma)$. We will break this proof into two steps:

(a) ($\subseteq$) Take $\ell \in \pi \cdot (\mathsf{dom}(\sigma') \setminus \mathsf{dom}(\sigma))$. Then $\ell = \pi(\ell')$ for some $\ell' \in \mathsf{dom}(\sigma') \setminus \mathsf{dom}(\sigma)$. Moreover, we have $\ell' \mapsto v \in \sigma' \setminus \sigma$ for some $v \in \mathsf{Value}$. This implies both that $\ell' \mapsto v \in \sigma'$ and $\ell' \mapsto v \notin \sigma$. Thus, by definition, $\ell \mapsto \pi \cdot v \in \pi \cdot \sigma'$. To see why $\ell \mapsto \pi \cdot v \notin \pi \cdot \sigma$, assume that $\ell \mapsto \pi \cdot v \in \pi \cdot \sigma$: because $\pi^{-1}(\ell) = \ell'$, this would directly imply $\ell' \mapsto v \in \sigma$, which is impossible. We then have that $\ell \mapsto \pi \cdot v \in (\pi \cdot \sigma') \setminus (\pi \cdot \sigma)$, meaning that $\ell \in \mathsf{dom}(\pi \cdot \sigma') \setminus \mathsf{dom}(\pi \cdot \sigma)$. Therefore, $\pi \cdot (\mathsf{dom}(\sigma') \setminus \mathsf{dom}(\sigma)) \subseteq \mathsf{dom}(\pi \cdot \sigma') \setminus \mathsf{dom}(\pi \cdot \sigma)$.

(b) ($\supseteq$) Take $\ell \in \mathsf{dom}(\pi \cdot \sigma') \setminus \mathsf{dom}(\pi \cdot \sigma)$. Then $\ell \mapsto v \in (\pi \cdot \sigma') \setminus (\pi \cdot \sigma)$ for some $v \in \mathsf{Value}$. This directly implies that $\ell \mapsto v \in (\pi \cdot \sigma')$ and $\ell \mapsto v \notin (\pi \cdot \sigma)$. Say that $\ell' = \pi^{-1}(\ell)$. Then, by definition $\ell' \mapsto \pi^{-1} \cdot v \in \sigma'$. To see why $\ell' \mapsto \pi^{-1} \cdot v \notin \sigma$, notice that, like the previous case, $\ell' \mapsto \pi^{-1} \cdot v \in \sigma$ would imply a contradiction. Therefore, we have that $\ell' \mapsto \pi^{-1} \cdot v \in \sigma' \setminus \sigma$, meaning $\ell' \in \mathsf{dom}(\sigma') \setminus \mathsf{dom}(\sigma)$. Hence, $\ell \in \pi \cdot (\mathsf{dom}(\sigma') \setminus \mathsf{dom}(\sigma))$, and we conclude $\pi \cdot (\mathsf{dom}(\sigma') \setminus \mathsf{dom}(\sigma)) \supseteq \mathsf{dom}(\pi \cdot \sigma') \setminus \mathsf{dom}(\pi \cdot \sigma)$.

Therefore, $\pi \cdot (\mathsf{dom}(\sigma') \setminus \mathsf{dom}(\sigma)) = \mathsf{dom}(\pi \cdot \sigma') \setminus \mathsf{dom}(\pi \cdot \sigma)$, and we are done.

Therefore, $\pi \cdot \mathsf{run}(\rho, e, \sigma) = \mathsf{run}(\pi \cdot \rho, e, \pi \cdot \sigma)$. □

Lemma B.17 (Nominal determinism)**.** If $(\rho, e, \sigma) \in \mathsf{Config}$ and $\rho, s \vdash (e, \sigma) \Downarrow (v_1, \sigma_1)$ and $\rho, s \vdash (e, \sigma) \Downarrow (v_2, \sigma_2)$ then $\langle \mathsf{dom}(\sigma_1) \setminus \mathsf{dom}(\sigma)\rangle(v_1, \sigma_1) = \langle \mathsf{dom}(\sigma_2) \setminus \mathsf{dom}(\sigma)\rangle(v_2, \sigma_2)$.

Proof. By induction on $\rho, s \vdash (e, \sigma) \Downarrow (v_1, \sigma_1)$ and inversion on $\rho, s \vdash (e, \sigma) \Downarrow (v_2, \sigma_2)$. We show selected cases:

- (Ref) Suppose $\rho, s \vdash (\mathtt{ref}\, x, \sigma) \Downarrow (\ell, \sigma \uplus \{\ell \mapsto \rho(x)\})$ and $\rho, s \vdash (\mathtt{ref}\, x, \sigma) \Downarrow (\ell', \sigma \uplus \{\ell' \mapsto \rho(x)\})$. Then $\langle \ell \rangle(\ell, \sigma \uplus \{\ell \mapsto \rho(x)\}) = \langle \ell' \rangle(\ell', \sigma \uplus \{\ell' \mapsto \rho(x)\})$.
- (Let) Suppose

(1) $\rho, s_1 \vdash (e_1, \sigma) \Downarrow (v_1, \sigma_1)$
(2) $\rho[x \mapsto v_1], s_2 \vdash (e_2, \sigma_1) \Downarrow (v_2, \sigma_2)$
(3) $\rho, s_1 \vdash (e_1, \sigma) \Downarrow (v_1', \sigma_1')$
(4) $\rho[x \mapsto v_1'], s_2 \vdash (e_2, \sigma_1') \Downarrow (v_2', \sigma_2')$

By IH on (1) and (3), $\langle \sigma_1 \setminus \sigma \rangle(v_1, \sigma_1) = \langle \sigma_1' \setminus \sigma \rangle(v_1', \sigma_1')$, so there are products of transpositions π, π' swapping $\sigma_1 \setminus \sigma$ and $\sigma_1' \setminus \sigma$ with some fresh set of locations F respectively such that $\pi \cdot (v_1, \sigma_1) = \pi' \cdot (v_1', \sigma_1')$. (Note we have abused notation and written things like $\sigma_2 \setminus \sigma_1$ instead of $\mathsf{dom}(\sigma_2) \setminus \mathsf{dom}(\sigma_1)$ inside of $\langle - \rangle$; we will continue to do this throughout for legibility's sake.) By Theorem B.15, (4) implies $\rho[x \mapsto v_1], s_2 \vdash (e_2, \sigma_1) \Downarrow \pi^{-1}\pi' \cdot (v_2', \sigma_2')$, which by IH with (2) implies $\langle \sigma_2 \setminus \sigma_1 \rangle(v_2, \sigma_2) = \langle \pi^{-1}\pi' \sigma_2' \setminus \sigma_1 \rangle(\pi^{-1}\pi' \cdot (v_2', \sigma_2')) = \pi^{-1}\pi' \cdot \langle \sigma_2' \setminus \sigma_1' \rangle(v_2', \sigma_2')$. Rearranging gives $\langle \sigma_2 \setminus \sigma_1 \rangle(\pi \cdot (v_2, \sigma_2)) = \langle \sigma_2' \setminus \sigma_1' \rangle(\pi' \cdot (v_2', \sigma_2'))$, where π, π' are allowed to be brought under $\langle - \rangle$ because $(\sigma_1 \setminus \sigma) \cup F$ is disjoint from $\sigma_2 \setminus \sigma_1$ and $(\sigma_1' \setminus \sigma) \cup F$ is disjoint from $\sigma_2' \setminus \sigma_1'$. This implies there exist products of transpositions p, p' swapping $\sigma_2 \setminus \sigma_1, \sigma_2' \setminus \sigma_1'$ with some fresh set of locations F' respectively such that $p \cdot \pi \cdot (v_2, \sigma_2) = p' \cdot \pi' \cdot (v_2', \sigma_2')$. Since p, π are disjoint products of transpositions, $p\pi$ is itself a product of transpositions swapping $\sigma_2 \setminus \sigma$ with $F \uplus F'$; analogously, $p'\pi'$ swaps $\sigma_2' \setminus \sigma$ with $F \uplus F'$. This establishes $\langle \sigma_2 \setminus \sigma \rangle(v_2, \sigma_2) = \langle \sigma_2' \setminus \sigma \rangle(v_2', \sigma_2')$, as needed.

All other cases are standard. □

Definition B.18. For $S \subseteq_{\mathsf{fin}} \mathsf{SymVar}$, let $\widehat{A^S}$ be the set of functions $\mathsf{Model}_S \to A_\perp$ with finite image. We will implicitly coerce elements of $\widehat{A^S}$ into elements of $\widehat{A}$, and into elements of $\widehat{A^T}$ for $T \supseteq S$.

Definition B.19. For $\widehat{a} \in \widehat{A^{S \uplus T}}$ and $m \in \mathsf{Model}_S$, define $\widehat{a}|_m \in \widehat{A^T}$ by $\widehat{a}|_m(m') = \widehat{a}(m \uplus m')$.

Definition B.20. For a store σ, symbolic value $\widehat{v}$, and symbolic store $\widehat{\sigma}'$ with $\mathsf{symvars}(\widehat{v}, \widehat{\sigma}') \subseteq V$, let $\mathsf{result}_V(\sigma, \widehat{v}, \widehat{\sigma}')$ be the function

$$\mathsf{result}_V(\sigma, \widehat{v}, \widehat{\sigma}') : \mathsf{Model}_V \to \mathsf{Result}_\perp$$

$$\mathsf{result}_V(\sigma, \widehat{v}, \widehat{\sigma}')(m) = \begin{cases} \langle \mathsf{dom}(\widehat{\sigma}'(m)) \setminus \mathsf{dom}(\sigma) \rangle(\widehat{v}(m), \widehat{\sigma}'(m)) & \text{if } \widehat{v}(m) \neq \perp, \widehat{\sigma}'(m) \neq \perp \\ \perp & \text{otherwise} \end{cases}$$

Theorem B.21 (Correctness of Idealized Rosette)**.** Suppose that $\widehat{\rho} \vdash (e, \widehat{\sigma}) \Downarrow\!\Downarrow (\widehat{v}, \widehat{\sigma}', \psi)$. Then, for every $m \in \mathsf{Model}_{(\mathsf{symvars}\, \widehat{\rho}, \widehat{\sigma})}$ with $(\widehat{\rho}(m), e, \widehat{\sigma}(m)) \in \mathsf{Config}$ and $V = \mathsf{symvars}(\widehat{v}, \widehat{\sigma}') \setminus \mathsf{symvars}(\widehat{\rho}, \widehat{\sigma})$:

(1) The following functions have the same image:

$$\mathsf{run}(\widehat{\rho}(m), e, \widehat{\sigma}(m)) : \mathsf{Bool}^{\mathbb{N}} \to \mathsf{Result}_\perp$$
$$\mathsf{result}_V(\widehat{\sigma}(m), \widehat{v}|_m, \widehat{\sigma}'|_m) : \mathsf{Model}_V \to \mathsf{Result}_\perp$$

(2) $\psi(m \uplus m') = \mathsf{T}$ if and only if $\widehat{v}(m \uplus m') \neq \perp$ and $\widehat{\sigma}'(m \uplus m') \neq \perp$ for all $m' \in \mathsf{Model}_V$

Proof. Let $\widehat{\rho} \in \widehat{\mathsf{Env}}$, $e \in \mathsf{Expr}$, and $\widehat{\sigma} \in \widehat{\mathsf{Store}}$. Additionally, let $\widehat{v} \in \widehat{\mathsf{Value}}$, $\widehat{\sigma}' \in \widehat{\mathsf{Store}}$, and $\psi : \mathsf{Model}_V \to \mathbb{B}$ for $V = \mathsf{symvars}(\widehat{v}, \widehat{\sigma}') \setminus \mathsf{symvars}(\widehat{\rho}, \widehat{\sigma})$ such that

$$\widehat{\rho} \vdash (e, \widehat{\sigma}) \Downarrow\!\Downarrow (\widehat{v}, \widehat{\sigma}', \psi)$$

We will first prove (1) and (2) simulatenously by structural induction on $\widehat{\rho} \vdash (e, \widehat{\sigma}) \Downarrow\!\Downarrow (\widehat{v}, \widehat{\sigma}', \psi)$. Let the inductive hypothesis $P(\widehat{\rho}, e, \widehat{\sigma}, \widehat{v}, \widehat{\sigma}', \psi)$ be that, if $\widehat{\rho} \vdash (e, \widehat{\sigma}) \Downarrow\!\Downarrow (\widehat{v}, \widehat{\sigma}', \psi)$, the following two functions have the same image for all $m \in \mathsf{Model}_{\mathsf{symvars}(\widehat{\rho}, \widehat{\sigma})}$ with $(\widehat{\rho}(m), e, \widehat{\sigma}(m)) \in \mathsf{Config}$:

$$\mathsf{run}(\widehat{\rho}(m), e, \widehat{\sigma}(m)) : \mathbb{B}^{\mathbb{N}} \to \mathsf{Result}_\perp$$
$$\mathsf{result}_V(\widehat{\sigma}(m), \widehat{v}|_m, \widehat{\sigma}'|_m) : \mathsf{Model}_V \to \mathsf{Result}_\perp$$

and

$$\psi(m \uplus m') = \mathsf{T} \iff \widehat{v}(m \uplus m') \neq \perp \text{ and } \widehat{\sigma}'(m \uplus m') \neq \perp$$

for all $m' \in \mathsf{Model}_V$.

Let $m \in \mathsf{Model}_{\mathsf{symvars}\, \widehat{\rho}, \widehat{\sigma}}$ with $(\widehat{\rho}(m), e, \widehat{\sigma}(m)) \in \mathsf{Config}$. We have several cases:

(Var) Consider $e = x$. The only abstract big-step judgement whose conclusion matches $e = x$ is Var. Hence, by inversion of our assumption that $\widehat{\rho} \vdash (e, \widehat{\sigma}) \Downarrow\!\Downarrow (\widehat{v}, \widehat{\sigma}', \psi)$, we get

$$\widehat{\rho} \vdash (x, \widehat{\sigma}) \Downarrow\!\Downarrow (\widehat{\rho}(x), \widehat{\sigma}, \mathsf{T})$$

We will begin by showing (1). It suffices to show that the following two functions have the same image

$$\mathsf{run}(\widehat{\rho}(m), x, \widehat{\sigma}(m)) : \mathbb{B}^{\mathbb{N}} \to \mathsf{Result}_{\bot}$$
$$\mathsf{result}_V(\widehat{\sigma}(m), \widehat{\rho}(x)|_m, \widehat{\sigma}|_m) : \mathsf{Model}_V \to \mathsf{Result}_{\bot}$$

By the assumption that $(\widehat{\rho}(m), x, \widehat{\sigma}(m)) \in \mathsf{Config}$, we have that $\mathsf{locs}(\widehat{\rho}(m)) \cup \mathsf{locs}(\widehat{\sigma}(m)) \subseteq \mathsf{dom}(\widehat{\sigma}(m))$, $\mathsf{FV}(\widehat{\rho}(m)(y)) = \varnothing$ for all $y \in \mathsf{dom}(\widehat{\rho}(m))$, and $\mathsf{FV}(e) \subseteq \mathsf{dom}(\widehat{\rho}(m))$. This implies that $x \in \mathsf{dom}(\widehat{\rho}(m))$, meaning further that $\widehat{\rho}(m)(x) \neq \bot$. These facts allow us to prove, for any $s \in \mathbb{B}^{\mathbb{N}}$, that

$$\widehat{\rho}(m), s \vdash (x, \widehat{\sigma}(m)) \Downarrow (\widehat{\rho}(m)(x), \widehat{\sigma}(m))$$

using the concrete big-step judgement Var. Then, by definition of run, we have that

$$\mathsf{run}(\widehat{\rho}(m), x, \widehat{\sigma}(m))(s) = \langle\varnothing\rangle(\widehat{\rho}(m)(x), \widehat{\sigma}(m))$$

for any $s \in \mathbb{B}^{\mathbb{N}}$ due to the fact that $\mathsf{dom}(\widehat{\sigma}(m)) \setminus \mathsf{dom}(\widehat{\sigma}(m)) = \varnothing$. Thus,

$$\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), x, \widehat{\sigma}(m))) = \{\langle\varnothing\rangle(\widehat{\rho}(m)(x), \widehat{\sigma}(m))\}$$

Now, observe that $V = \mathsf{symvars}(\widehat{\rho}(m)(x), \widehat{\sigma}) \setminus \mathsf{symvars}(\widehat{\rho}, \widehat{\sigma}) = \varnothing$. Thus, we get that $\mathsf{Model}_V = \{\varnothing\}$. Note the following two equivalences:

$$\widehat{\rho}(x)|_m(\varnothing) = \widehat{\rho}(m \uplus \varnothing)(x) = \widehat{\rho}(m)(x)$$
$$\widehat{\sigma}|_m(\varnothing) = \widehat{\sigma}(m \uplus \varnothing) = \widehat{\sigma}(m)$$

Hence, we have, by definition, that

$$\mathsf{result}_V(\widehat{\sigma}(m), \widehat{\rho}(x)|_m, \widehat{\sigma}|_m)(\varnothing) = \langle\varnothing\rangle(\widehat{\rho}(m)(x), \widehat{\sigma}(m))$$

due to the fact that $\mathsf{dom}(\widehat{\sigma}(m)) \setminus \mathsf{dom}(\widehat{\sigma}(m)) = \varnothing$. This means that

$$\mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), \widehat{\rho}(x)|_m, \widehat{\sigma}|_m)) = \{\langle\varnothing\rangle(\widehat{\rho}(m)(x), \widehat{\sigma}(m))\}$$

Thus, $\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), x, \widehat{\sigma}(m))) = \mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), \widehat{\rho}(x)|_m, \widehat{\sigma}|_m))$, completing our proof of (1).

To prove (2), it suffices to show that

$$\mathsf{T}(m \uplus m') = \mathsf{T} \iff \widehat{\rho}(m \uplus m')(x) \neq \bot \text{ and } \widehat{\sigma}(m \uplus m') \neq \bot$$

for all $m' \in \mathsf{Model}_V$. Notice first that

$$\mathsf{T}(m \uplus m') = \mathsf{T}$$

for any $m' \in \mathsf{Model}_V$. Likewise, because $\bot \notin \mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), \widehat{\rho}(x)|_m, \widehat{\sigma}|_m))$ (shown in the proof of (1)), we know that $\widehat{\rho}(x)|_m(m') \neq \bot$ and $\widehat{\sigma}|_m(m') \neq \bot$ for all $m' \in \mathsf{Model}_V$ by definition of result_V. Hence, in all cases, $\mathsf{T}(m \uplus m') = \mathsf{T}$, $\widehat{\rho}(m \uplus m')(x) \neq \bot$, and $\widehat{\sigma}(m \uplus m') \neq \bot$. This completes our proof of (2).

(True) Consider $e = \mathtt{true}$. The only abstract big-step judgement whose conclusion matches $e = \mathtt{true}$ is True. Hence, by inversion of our assumption that $\widehat{\rho} \vdash (e, \widehat{\sigma}) \Downarrow\!\Downarrow (\widehat{v}, \widehat{\sigma}', \psi)$, we get

$$\widehat{\rho} \vdash (\mathtt{true}, \widehat{\sigma}) \Downarrow\!\Downarrow ([\mathsf{T} : \mathtt{true}], \widehat{\sigma}, \mathsf{T})$$

We will begin by showing (1). It suffices to show that the following two functions have the same image

$$\mathsf{run}(\widehat{\rho}(m), \mathtt{true}, \widehat{\sigma}(m)) : \mathbb{B}^{\mathbb{N}} \to \mathsf{Result}_\perp$$
$$\mathsf{result}_V(\widehat{\sigma}(m), [\mathsf{T} : \mathtt{true}]|_m, \widehat{\sigma}|_m) : \mathsf{Model}_V \to \mathsf{Result}_\perp$$

By the assumption that $(\widehat{\rho}(m), e, \widehat{\sigma}(m)) \in \mathsf{Config}$, we know that $\widehat{\rho}(m) \neq \perp$ and $\widehat{\sigma}(m) \neq \perp$. Hence, we can prove, for any $s \in \mathbb{B}^{\mathbb{N}}$, that

$$\widehat{\rho}(m), s \vdash (\mathtt{true}, \widehat{\sigma}(m)) \Downarrow (\mathtt{true}, \widehat{\sigma}(m))$$

using the concrete big-step judgement True. Then, by definition of run, we have that

$$\mathsf{run}(\widehat{\rho}(m), \mathtt{true}, \widehat{\sigma}(m))(s) = \langle\varnothing\rangle(\mathtt{true}, \widehat{\sigma}(m))$$

for any $s \in \mathbb{B}^{\mathbb{N}}$ due to the fact that $\mathsf{dom}(\widehat{\sigma}(m)) \setminus \mathsf{dom}(\widehat{\sigma}(m)) = \varnothing$. Thus,

$$\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), \mathtt{true}, \widehat{\sigma}(m))) = \{\langle\varnothing\rangle(\mathtt{true}, \widehat{\sigma}(m))\}$$

Now, observe that $V = \mathsf{symvars}([\mathsf{T} : \mathtt{true}], \widehat{\sigma}) \setminus \mathsf{symvars}(\widehat{\rho}, \widehat{\sigma}) = \varnothing$. Thus, $\mathsf{Model}_V = \{\varnothing\}$. Note the following two equivalences:

$$[\mathsf{T} : \mathtt{true}]|_m(\varnothing) = [\mathsf{T} : \mathtt{true}](m \uplus \varnothing) = [\mathsf{T} : \mathtt{true}](m) = \mathtt{true}$$
$$\widehat{\sigma}|_m(\varnothing) = \widehat{\sigma}(m \uplus \varnothing) = \widehat{\sigma}(m)$$

Hence, we have, by definition, that

$$\mathsf{result}_V(\widehat{\sigma}(m), [\mathsf{T} : \mathtt{true}]|_m, \widehat{\sigma}|_m)(\varnothing) = \langle\varnothing\rangle(\mathtt{true}, \widehat{\sigma}(m))$$

due to the fact that $\mathsf{dom}(\widehat{\sigma}(m)) \setminus \mathsf{dom}(\widehat{\sigma}(m)) = \varnothing$. This means

$$\mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), [\mathsf{T} : \mathtt{true}]|_m, \widehat{\sigma}|_m)) = \{\langle\varnothing\rangle(\mathtt{true}, \widehat{\sigma}(m))\}$$

Thus, $\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), \mathtt{true}, \widehat{\sigma}(m))) = \mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), [\mathsf{T} : \mathtt{true}]|_m, \widehat{\sigma}|_m))$. This completes our proof of (1).

To prove (2), it suffices to show that

$$\mathsf{T}(m \uplus m') = \mathsf{T} \iff [\mathsf{T} : \mathtt{true}](m \uplus m') \neq \perp \text{ and } \widehat{\sigma}(m \uplus m') \neq \perp$$

for all $m' \in \mathsf{Model}_V$. Notice first that

$$\mathsf{T}(m \uplus m') = \mathsf{T}$$

for any $m' \in \mathsf{Model}_V$. Likewise, because $\perp \notin \mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), [\mathsf{T} : \mathtt{true}]|_m, \widehat{\sigma}|_m))$ (shown in the proof of (1)), we know that $[\mathsf{T} : \mathtt{true}]|_m(m') \neq \perp$ and $\widehat{\sigma}|_m(m') \neq \perp$ for all $m' \in \mathsf{Model}_V$ by definition of result_V. Hence, in all cases, $\mathsf{T}(m \uplus m') = \mathsf{T}$, $[\mathsf{T} : \mathtt{true}](m \uplus m') \neq \perp$, and $\widehat{\sigma}(m \uplus m') \neq \perp$. This completes our proof of (2).

(False) Consider $e = \mathtt{false}$. Apply identical reasoning as in the previous case.

(Num) Consider $e = r$ for $r \in \mathbb{Q}$. The only abstract big-step judgement whose conclusion matches $e = r$ for $r \in \mathbb{Q}$ is Num. Hence, by inversion of our assumption that $\widehat{\rho} \vdash (e, \widehat{\sigma}) \Downarrow\!\Downarrow (\widehat{v}, \widehat{\sigma}', \psi)$, we get

$$\widehat{\rho} \vdash (r, \widehat{\sigma}) \Downarrow\!\Downarrow ([\mathsf{T} : r], \widehat{\sigma}, \mathsf{T})$$

We will begin by showing (1). It suffices to show that the following two functions have the same image

$$\mathsf{run}(\widehat{\rho}(m), r, \widehat{\sigma}(m)) : \mathbb{B}^{\mathbb{N}} \to \mathsf{Result}_\perp$$
$$\mathsf{result}_V(\widehat{\sigma}(m), [\mathsf{T} : r]|_m, \widehat{\sigma}|_m) : \mathsf{Model}_V \to \mathsf{Result}_\perp$$

By the assumption that $(\widehat{\rho}(m), e, \widehat{\sigma}(m)) \in \text{Config}$, we know that $\widehat{\rho}(m) \neq \bot$ and $\widehat{\sigma}(m) \neq \bot$. Hence, we can prove, for any $s \in \mathbb{B}^{\mathbb{N}}$, that

$$\widehat{\rho}(m), s \vdash (r, \widehat{\sigma}(m)) \Downarrow (r, \widehat{\sigma}(m))$$

using the concrete big-step judgement Num. Then, by the definition of run, we have that

$$\mathsf{run}(\widehat{\rho}(m), r, \widehat{\sigma}(m))(s) = \langle\varnothing\rangle(r, \widehat{\sigma}(m))$$

for any $s \in \mathbb{B}^{\mathbb{N}}$ due to the fact that $\mathsf{dom}(\widehat{\sigma}(m)) \setminus \mathsf{dom}(\widehat{\sigma}(m)) = \varnothing$. Thus,

$$\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), r, \widehat{\sigma}(m))) = \{\langle\varnothing\rangle(r, \widehat{\sigma}(m))\}$$

Now, observe that $V = \mathsf{symvars}([\mathsf{T} : r], \widehat{\sigma}) \setminus \mathsf{symvars}(\widehat{\rho}, \widehat{\sigma}) = \varnothing$. Thus, $\mathsf{Model}_V = \{\varnothing\}$. Note the following two equivalences:

$$[\mathsf{T} : r]|_m(\varnothing) = [\mathsf{T} : r](m \uplus \varnothing) = [\mathsf{T} : r](m) = r$$
$$\widehat{\sigma}|_m(\varnothing) = \widehat{\sigma}(m \uplus \varnothing) = \widehat{\sigma}(m)$$

Hence, we have, by definition, that

$$\mathsf{result}_V(\widehat{\sigma}(m), [\mathsf{T} : r]|_m, \widehat{\sigma}|_m)(\varnothing) = \langle\varnothing\rangle(r, \widehat{\sigma}(m))$$

due to the fact that $\mathsf{dom}(\widehat{\sigma}(m)) \setminus \mathsf{dom}(\widehat{\sigma}(m)) = \varnothing$. This means

$$\mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), [\mathsf{T} : r]|_m, \widehat{\sigma}|_m)) = \{\langle\varnothing\rangle(r, \widehat{\sigma}(m))\}$$

Thus, $\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), r, \widehat{\sigma}(m))) = \mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), [\mathsf{T} : r]|_m, \widehat{\sigma}|_m))$. This completes our proof of (1).

To prove (2), it suffices to show that

$$\mathsf{T}(m \uplus m') = \mathsf{T} \iff [\mathsf{T} : r](m \uplus m') \neq \bot \text{ and } \widehat{\sigma}(m \uplus m') \neq \bot$$

for all $m' \in \mathsf{Model}_V$. Notice first that

$$\mathsf{T}(m \uplus m') = \mathsf{T}$$

for any $m' \in \mathsf{Model}_V$. Likewise, because $\bot \notin \mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), [\mathsf{T} : r]|_m, \widehat{\sigma}|_m))$ (shown in the proof of (1)), we know that $[\mathsf{T} : r]|_m(m') \neq \bot$ and $\widehat{\sigma}|_m(m') \neq \bot$ for all $m' \in \mathsf{Model}_V$ by definition of result_V. Hence, in all cases, $\mathsf{T}(m \uplus m') = \mathsf{T}$, $[\mathsf{T} : r](m \uplus m') \neq \bot$, and $\widehat{\sigma}(m \uplus m') \neq \bot$. This completes our proof of (2).

(Lam) Consider $e = \lambda x.e'$ for $e' \in \text{Expr}$. The only abstract big-step judgement whose conclusion matches $e = \lambda x.e'$ is Lam. Hence, by inversion of our assumption that $\widehat{\rho} \vdash (e, \widehat{\sigma}) \Downarrow\!\Downarrow (\widehat{v}, \widehat{\sigma}', \psi)$, we get

$$\widehat{\rho} \vdash (\lambda x.e', \widehat{\sigma}) \Downarrow\!\Downarrow ([\mathsf{T} : \mathtt{clo}(\lambda x.e', \widehat{\rho})], \widehat{\sigma}, \mathsf{T})$$

We will begin by showing (1). It suffices to show that the following two functions have the same image

$$\mathsf{run}(\widehat{\rho}(m), \lambda x.e', \widehat{\sigma}(m)) : \mathbb{B}^{\mathbb{N}} \to \mathsf{Result}_\bot$$
$$\mathsf{result}_V(\widehat{\sigma}(m), [\mathsf{T} : \mathtt{clo}(\lambda x.e', \widehat{\rho})]|_m, \widehat{\sigma}|_m) : \mathsf{Model}_V \to \mathsf{Result}_\bot$$

By the assumption that $(\widehat{\rho}(m), e, \widehat{\sigma}(m)) \in \text{Config}$, we know that $\widehat{\rho}(m) \neq \bot$ and $\widehat{\sigma}(m) \neq \bot$. Hence, we can prove, for any $s \in \mathbb{B}^{\mathbb{N}}$, that

$$\widehat{\rho}(m), s \vdash (\lambda x.e', \widehat{\sigma}(m)) \Downarrow (\mathtt{clo}(\lambda x.e', \widehat{\rho}(m)), \widehat{\sigma}(m))$$

using the concrete big-step judgement Lam. Then, by definition of run, we have that

$$\mathsf{run}(\widehat{\rho}(m), \lambda x.e', \widehat{\sigma}(m))(s) = \langle\varnothing\rangle(\mathtt{clo}(\lambda x.e', \widehat{\rho}(m)), \widehat{\sigma}(m))$$

for any $s \in \mathbb{B}^{\mathbb{N}}$ due to the fact that $\mathsf{dom}(\widehat{\sigma}(m)) \setminus \mathsf{dom}(\widehat{\sigma}(m)) = \varnothing$. Thus,

$$\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), \lambda x.e', \widehat{\sigma}(m))) = \{\langle\varnothing\rangle(\mathtt{clo}(\lambda x.e', \widehat{\rho}(m)), \widehat{\sigma}(m))\}$$

Now, observe that $V = \mathsf{symvars}([\mathsf{T} : \mathtt{clo}(\lambda x.e', \widehat{\rho})], \widehat{\sigma}) \setminus \mathsf{symvars}(\widehat{\rho}, \widehat{\sigma}) = \varnothing$. Thus, we get $\mathsf{Model}_V = \{\varnothing\}$. Note the following two equivalences:

$$[\mathsf{T} : \mathtt{clo}(\lambda x.e', \widehat{\rho})]|_m(\varnothing) = [\mathsf{T} : \mathtt{clo}(\lambda x.e', \widehat{\rho})](m \uplus \varnothing) = \mathtt{clo}(\lambda x.e', \widehat{\rho}(m))$$
$$\widehat{\sigma}|_m(\varnothing) = \widehat{\sigma}(m \uplus \varnothing) = \widehat{\sigma}(m)$$

noting that $\mathtt{clo}(\lambda x.e', \widehat{\rho})(m) = \mathtt{clo}(\lambda x.e', \widehat{\rho}(m))$ because $\widehat{\rho}(m) \neq \bot$. Hence, we have, by defintion, that

$$\mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), [\mathsf{T} : \mathtt{clo}(\lambda x.e', \widehat{\rho})]|_m, \widehat{\sigma}|_m)) = \{\langle\varnothing\rangle(\mathtt{clo}(\lambda x.e', \widehat{\rho}(m)), \widehat{\sigma}(m))\}$$

Thus, $\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), \lambda x.e', \widehat{\sigma}(m))) = \mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), [\mathsf{T} : \mathtt{clo}(\lambda x.e', \widehat{\rho})]|_m, \widehat{\sigma}|_m))$. This completes our proof of (1).
To prove (2), it suffices to show that

$$\mathsf{T}(m \uplus m') = \mathsf{T} \iff [\mathsf{T} : \mathtt{clo}(\lambda x.e', \widehat{\rho})](m \uplus m') \neq \bot \text{ and } \widehat{\sigma}(m \uplus m') \neq \bot$$

for all $m' \in \mathsf{Model}_V$. Notice first that

$$\mathsf{T}(m \uplus m') = \mathsf{T}$$

for any $m' \in \mathsf{Model}_V$. Likewise, because $\bot \notin \mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), [\mathsf{T} : \mathtt{clo}(\lambda x.e', \widehat{\rho})]|_m, \widehat{\sigma}|_m))$ (shown in the proof of (1)), we know that $[\mathsf{T} : \mathtt{clo}(\lambda x.e', \widehat{\rho})]|_m(m') \neq \bot$ and $\widehat{\sigma}|_m(m') \neq \bot$ for all $m' \in \mathsf{Model}_V$ by definition of result_V. Hence, in all cases, $\mathsf{T}(m \uplus m') = \mathsf{T}$, $[\mathsf{T} : \mathtt{clo}(\lambda x.e', \widehat{\rho})](m \uplus m') \neq \bot$, and $\widehat{\sigma}(m \uplus m') \neq \bot$. This completes our proof of (2).

(Arith) Consider $e = x \oplus y$ for $\oplus \in \{+, -, \times\}$. The only abstract big-step judgement whose conclusion matches $e = x \oplus y$ is Arith. Hence, by inversion of our assumption that $\widehat{\rho} \vdash (e, \widehat{\sigma}) \Downarrow (\widehat{v}, \widehat{\sigma}', \psi)$, we get

$$\widehat{\rho} \vdash ((x \oplus y), \widehat{\sigma}) \Downarrow ([\varphi_i \wedge \varphi_j : r_i \,[\![\oplus]\!]\, r_j]_{i \in I, j \in J}, \widehat{\sigma}, \bigvee_{i \in I, j \in J} (\varphi_i \wedge \varphi_j))$$

such that

$$\widehat{\rho}(x) = [\varphi_i : r_i]_{i \in I} \uplus_{\mathsf{Num}} \widehat{v}_1$$
$$\widehat{\rho}(y) = [\varphi_j : r_j]_{j \in J} \uplus_{\mathsf{Num}} \widehat{v}_2$$

where $r_i, r_j \in \mathbb{Q}$ for all $i \in I, j \in J$.
We will begin by showing (1). It suffices to show that the following two functions have the same image

$$\mathsf{run}(\widehat{\rho}(m), x \oplus y, \widehat{\sigma}(m)) : \mathbb{B}^{\mathbb{N}} \to \mathsf{Result}_\bot$$
$$\mathsf{result}_V(\widehat{\sigma}(m), [\varphi_i \wedge \varphi_j : r_i \,[\![\oplus]\!]\, r_j]_{i \in I, j \in J}|_m, \widehat{\sigma}|_m) : \mathsf{Model}_V \to \mathsf{Result}_\bot$$

There are two cases:

(a) Suppose $\varphi_i(m) = \mathsf{T}$ for some $i \in I$ and $\varphi_j(m) = \mathsf{T}$ for some $j \in J$. Then we get that

$$\widehat{\rho}(m)(x) = ([\varphi_i : r_i]_{i \in I} \uplus_{\mathsf{Num}} \widehat{v}_1)(m) = r_i$$
$$\widehat{\rho}(m)(y) = ([\varphi_i : r_j]_{j \in J} \uplus_{\mathsf{Num}} \widehat{v}_2)(m) = r_j$$

where $r_i, r_j \in \mathbb{Q}$. Hence, we can prove, for all $s \in \mathbb{B}^{\mathbb{N}}$, that

$$\widehat{\rho}(m), s \vdash (x \oplus y, \widehat{\sigma}(m)) \Downarrow (\widehat{\rho}(m)(x) \,[\![\oplus]\!]\, \widehat{\rho}(m)(y), \widehat{\sigma}(m))$$

using the concrete big-step judgement Arith. Therefore, for all $s \in \mathbb{B}^{\mathbb{N}}$, we have

$$\begin{aligned}\mathsf{run}(\widehat{\rho}(m), x \oplus y, \widehat{\sigma}(m))(s) &= \langle\varnothing\rangle(\widehat{\rho}(m)(x) \mathbin{[\![\oplus]\!]} \widehat{\rho}(m)(y), \widehat{\sigma}(m)) \\ &= \langle\varnothing\rangle(r_i \mathbin{[\![\oplus]\!]} r_j, \widehat{\sigma}(m))\end{aligned}$$

using the fact that $\mathsf{dom}(\widehat{\sigma}(m)) \setminus \mathsf{dom}(\widehat{\sigma}(m)) = \varnothing$. Therefore, we have that

$$\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), x \oplus y, \widehat{\sigma}(m))) = \{\langle\varnothing\rangle(r_i \mathbin{[\![\oplus]\!]} r_j, \widehat{\sigma}(m))\}$$

Now, observe that, for all $m' \in \mathsf{Model}_V$,

$$[\varphi_i \wedge \varphi_j : r_i \mathbin{[\![\oplus]\!]} r_j]_{i\in I, j\in J}|_m(m') = [\varphi_i \wedge \varphi_j : r_i \mathbin{[\![\oplus]\!]} r_j]_{i\in I, j\in J}(m \uplus m') = r_i \mathbin{[\![\oplus]\!]} r_j$$

meaning that, for all $m' \in \mathsf{Model}_V$,

$$\mathsf{result}_V(\widehat{\sigma}(m), [\varphi_i \wedge \varphi_j : r_i \mathbin{[\![\oplus]\!]} r_j]_{i\in I, j\in J}|_m, \widehat{\sigma}|_m)(m') = \langle\varnothing\rangle(r_i \mathbin{[\![\oplus]\!]} r_j, \widehat{\sigma}(m))$$

due to the fact that $\mathsf{dom}(\widehat{\sigma}(m)) \setminus \mathsf{dom}(\widehat{\sigma}(m)) = \varnothing$. Therefore, we have that

$$\mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), [\varphi_i \wedge \varphi_j : r_i \mathbin{[\![\oplus]\!]} r_j]_{i\in I, j\in J}|_m, \widehat{\sigma}|_m)) = \{\langle\varnothing\rangle(r_i \mathbin{[\![\oplus]\!]} r_j, \widehat{\sigma}(m))\}$$

Hence,

$$\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), x \oplus y, \widehat{\sigma}(m))) = \mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), [\varphi_i \wedge \varphi_j : r_i \mathbin{[\![\oplus]\!]} r_j]_{i\in I, j\in J}|_m, \widehat{\sigma}|_m))$$

(b) Suppose that at least one of (i) $\varphi_i(m) = \mathsf{F}$ for all $i \in I$ or (ii) $\varphi_j(m) = \mathsf{F}$ for all $j \in J$. It follows that $\mathsf{run}(\widehat{\rho}(m), x \oplus y, \widehat{\sigma}(m))(s) = \bot$ for all $s \in \mathbb{B}^{\mathbb{N}}$.
To make this argument precise, assume for the purposes of contradiction that

$$\mathsf{run}(\widehat{\rho}(m), x \oplus y, \widehat{\sigma}(m))(s) \neq \bot$$

for some $s \in \mathbb{B}^{\mathbb{N}}$. Then it must be the case that

$$\mathsf{run}(\widehat{\rho}(m), x \oplus y, \widehat{\sigma}(m))(s) = \langle L\rangle(v, \sigma)$$

for some $(v, \sigma) \in \mathsf{Value} \times \mathsf{Store}$. This immediately implies that $\widehat{\rho}(m)(x), \widehat{\rho}(m)(y) \in \mathbb{Q}$, which is impossible. If $\varphi_i(m) = \mathsf{F}$ for all $i \in I$, then $\widehat{\rho}(m)(x) \notin \mathbb{Q}$ by construction. Likewise, if $\varphi_j(m) = \mathsf{F}$ for all $j \in J$, then $\widehat{\rho}(m)(y) \notin \mathbb{Q}$ by construction. We conclude that

$$\mathsf{run}(\widehat{\rho}(m), x \oplus y, \widehat{\sigma}(m))(s) = \bot$$

for all $s \in \mathbb{B}^{\mathbb{N}}$. Hence,

$$\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), x \oplus y, \widehat{\sigma}(m))) = \{\bot\}$$

Now, following from our assumption that at least one of (i) $\varphi_i(m) = \mathsf{F}$ for all $i \in I$ or (ii) $\varphi_j(m) = \mathsf{F}$ for all $j \in J$, we have that

$$[\varphi_i \wedge \varphi_j : r_i \mathbin{[\![\oplus]\!]} r_j]_{i\in I, j\in J}|_m(m') = [\varphi_i \wedge \varphi_j : r_i \mathbin{[\![\oplus]\!]} r_j]_{i\in I, j\in J}(m \uplus m') = \bot$$

for all $m' \in \mathsf{Model}_V$. Therefore, for all $m' \in \mathsf{Model}_V$, we have that

$$\mathsf{result}_V(\widehat{\sigma}(m), [\varphi_i \wedge \varphi_j : r_i \mathbin{[\![\oplus]\!]} r_j]_{i\in I, j\in J}|_m, \widehat{\sigma}|_m)(m') = \bot$$

meaning that

$$\mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), [\varphi_i \wedge \varphi_j : r_i \mathbin{[\![\oplus]\!]} r_j]_{i\in I, j\in J}|_m, \widehat{\sigma}|_m)) = \{\bot\}$$

Hence,

$$\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), x \oplus y, \widehat{\sigma}(m))) = \mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), [\varphi_i \wedge \varphi_j : r_i \mathbin{[\![\oplus]\!]} r_j]_{i\in I, j\in J}|_m, \widehat{\sigma}|_m))$$

Therefore, $\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), x \oplus y, \widehat{\sigma}(m))) = \mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), [\varphi_i \wedge \varphi_j : r_i \,[\![\oplus]\!]\, r_j]_{i\in I, j\in J}|_m, \widehat{\sigma}|_m))$. This completes our proof of (1).

To prove (2), it suffices to show that

$$\left(\bigvee_{i\in I, j\in J} (\varphi_i \wedge \varphi_j)\right)(m \uplus m') = \mathsf{T} \iff [\varphi_i \wedge \varphi_j : r_i \,[\![\oplus]\!]\, r_j]_{i\in I, j\in J}(m \uplus m') \neq \bot \text{ and } \widehat{\sigma}(m \uplus m') \neq \bot$$

for all $m' \in \mathsf{Model}_V$.

($\Longrightarrow$) Suppose that $\bigvee_{i\in I, j\in J}(\varphi_i \wedge \varphi_j)(m \uplus m') = \mathsf{T}$. Then $(\varphi_i \wedge \varphi_j)(m) = \mathsf{T}$ for some $i \in I$, $j \in J$. Thus,

$$[\varphi_i \wedge \varphi_j : r_i \,[\![\oplus]\!]\, r_j]_{i\in I, j\in J}(m \uplus m') = r_i \,[\![\oplus]\!]\, r_j$$
$$\widehat{\sigma}(m \uplus m') = \widehat{\sigma}(m)$$

as desired.

($\Longleftarrow$) Suppose that $[\varphi_i \wedge \varphi_j : r_i \,[\![\oplus]\!]\, r_j]_{i\in I, j\in J}(m \uplus m') \neq \bot$ and $\widehat{\sigma}(m \uplus m') \neq \bot$. Then, by construction, it must be the case that $(\varphi_i \wedge \varphi_j)(m \uplus m') = \mathsf{T}$ for some $i \in I$ and $j \in I$. Therefore,

$$\left(\bigvee_{i\in I, j\in J} (\varphi_i \wedge \varphi_j)\right)(m \uplus m') = \mathsf{T}$$

as desired.

(Pair) Consider $e = (x, y)$. The only abstract big-step judgement whose conclusion matches $e = (x, y)$ is Pair. Hence, by inversion of our assumption that $\widehat{\rho} \vdash (e, \widehat{\sigma}) \Downarrow (\widehat{v}, \widehat{\sigma}', \psi)$, we get

$$\widehat{\rho} \vdash ((x, y), \widehat{\sigma}) \Downarrow ((\widehat{\rho}(x), \widehat{\rho}(y)), \widehat{\sigma}, \mathsf{T})$$

We will begin by showing (1). It suffices to show that the following two functions have the same image

$$\mathsf{run}(\widehat{\rho}(m), (x, y), \widehat{\sigma}(m)) : \mathbb{B}^{\mathbb{N}} \to \mathsf{Result}_\bot$$
$$\mathsf{result}_V(\widehat{\sigma}(m), (\widehat{\rho}(x), \widehat{\rho}(y))|_m, \widehat{\sigma}|_m) : \mathsf{Model}_V \to \mathsf{Result}_\bot$$

By the assumption that $(\widehat{\rho}(m), (x, y), \widehat{\sigma}(m)) \in \mathsf{Config}$, we know that $\mathsf{FV}((x, y)) \in \mathsf{dom}(\widehat{\rho}(m))$, meaning that $\widehat{\rho}(m)(x) \neq \bot$ and $\widehat{\rho}(m)(y) \neq \bot$. Therefore, we can prove, for any $s \in \mathbb{B}^{\mathbb{N}}$, that

$$\widehat{\rho}(m), s \vdash ((x, y), \widehat{\sigma}(m)) \Downarrow ((\widehat{\rho}(m)(x), \widehat{\rho}(m)(y)), \widehat{\sigma}(m))$$

using the concrete big-step judgement Pair. This implies, for all $s \in \mathbb{B}^{\mathbb{N}}$, that

$$\mathsf{run}(\widehat{\rho}(m), (x, y), \widehat{\sigma}(m))(s) = \langle\varnothing\rangle((\widehat{\rho}(m)(x), \widehat{\rho}(m)(y)), \widehat{\sigma}(m))$$

due to the fact that $\mathsf{dom}(\widehat{\sigma}(m)) \setminus \mathsf{dom}(\widehat{\sigma}(m)) = \varnothing$. Hence,

$$\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), (x, y), \widehat{\sigma}(m))) = \{\langle\varnothing\rangle((\widehat{\rho}(m)(x), \widehat{\rho}(m)(y)), \widehat{\sigma}(m))\}$$

Now, observe we have, for all $m' \in \mathsf{Model}_V$, that

$$(\widehat{\rho}(x), \widehat{\rho}(y))|_m(m') = (\widehat{\rho}(x), \widehat{\rho}(y))(m \uplus m') = (\widehat{\rho}(m)(x), \widehat{\rho}(m)(y)) \neq \bot$$
$$\widehat{\sigma}|_m(m') = \widehat{\sigma}(m \uplus m') = \widehat{\sigma}(m) \neq \bot$$

meaning that

$$\mathsf{result}_V(\widehat{\sigma}(m), (\widehat{\rho}(x), \widehat{\rho}(y))|_m, \widehat{\sigma}|_m)(m') = \langle\varnothing\rangle((\widehat{\rho}(m)(x), \widehat{\rho}(m)(y)), \widehat{\sigma}(m))$$

due to the fact that $\mathsf{dom}(\widehat{\sigma}(m)) \setminus \mathsf{dom}(\widehat{\sigma}(m)) = \varnothing$. Thus,

$$\mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), (\widehat{\rho}(x), \widehat{\rho}(y))|_m, \widehat{\sigma}|_m)) = \{\langle\varnothing\rangle((\widehat{\rho}(m)(x), \widehat{\rho}(m)(y)), \widehat{\sigma}(m))\}$$

so $\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), (x, y), \widehat{\sigma}(m))) = \mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), (\widehat{\rho}(x), \widehat{\rho}(y))|_m, \widehat{\sigma}|_m))$. This completes our proof of (1).

We now turn our attention to (2). It suffices to show that

$$\mathsf{T}(m \uplus m') = \mathsf{T} \iff (\widehat{\rho}(x), \widehat{\rho}(y))(m \uplus m') \neq \bot \text{ and } \widehat{\sigma}(m \uplus m') \neq \bot$$

for all $m' \in \mathsf{Model}_V$. Take any $m' \in \mathsf{Model}_V$. Clearly, $\mathsf{T}(m \uplus m') = \mathsf{T}$. We showed above that $(\widehat{\rho}(x), \widehat{\rho}(y))(m \uplus m') \neq \bot$ and $\widehat{\sigma}(m \uplus m') \neq \bot$ for all $m' \in \mathsf{Model}_V$. Therefore, we have that $\mathsf{T}(m \uplus m') = \mathsf{T}$, that $(\widehat{\rho}(x), \widehat{\rho}(y))(m \uplus m') \neq \bot$, and that $\widehat{\sigma}(m \uplus m') \neq \bot$ in all cases. This proves (2).

(Fst) Consider $e = \mathtt{fst}\, x$. The only abstract big-step judgement whose conclusion matches $e = \mathtt{fst}\, x$ is Fst. Hence, by inversion of our assumption that $\widehat{\rho} \vdash (e, \widehat{\sigma}) \Downarrow (\widehat{v}, \widehat{\sigma}', \psi)$, we get

$$\widehat{\rho} \vdash (\mathtt{fst}\, x, \widehat{\sigma}) \Downarrow ([\varphi_i : v_i]_{i \in I}, \widehat{\sigma}, \bigvee_{i \in I} \varphi_i)$$

such that $\widehat{\rho}(x) = [\varphi_i : (v_i, w_i)]_{i \in I} \uplus_{\mathsf{Pair}} \widehat{v}$.

We will begin by showing (1). It suffices to show that the following two functions have the same image

$$\mathsf{run}(\widehat{\rho}(m), \mathtt{fst}\, x, \widehat{\sigma}(m)) : \mathbb{B}^{\mathbb{N}} \to \mathsf{Result}_\bot$$
$$\mathsf{result}_V(\widehat{\sigma}(m), [\varphi_i : v_i]_{i \in I}|_m, \widehat{\sigma}|_m) : \mathsf{Model}_V \to \mathsf{Result}_\bot$$

There are two cases:

(a) Suppose that $\varphi_i(m) = \mathsf{T}$ for some $i \in I$. Then we have

$$\widehat{\rho}(m)(x) = ([\varphi_i : (v_i, w_i)]_{i \in I} \uplus_{\mathsf{Pair}} \widehat{v})(m) = (v_i, w_i)$$

Hence, we can prove, for all $s \in \mathbb{B}^{\mathbb{N}}$ that

$$\widehat{\rho}(m), s \vdash (\mathtt{fst}\, x, \widehat{\sigma}(m)) \Downarrow (v_i, \widehat{\sigma}(m))$$

using the concrete big-step judgement Fst. This implies that

$$\mathsf{run}(\widehat{\rho}(m), \mathtt{fst}\, x, \widehat{\sigma}(m))(s) = \langle \varnothing \rangle (v_i, \widehat{\sigma}(m))$$

for all $s \in \mathbb{B}^{\mathbb{N}}$, using the fact that $\mathsf{dom}(\widehat{\sigma}(m)) \setminus \mathsf{dom}(\widehat{\sigma}(m)) = \varnothing$. Hence,

$$\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), \mathtt{fst}\, x, \widehat{\sigma}(m))) = \{\langle \varnothing \rangle (v_i, \widehat{\sigma}(m))\}$$

Now, observe, for all $m' \in \mathsf{Model}_V$, we have that

$$[\varphi_i : v_i]_{i \in I}|_m(m') = [\varphi_i : v_i]_{i \in I}(m \uplus m') = v_i$$
$$\widehat{\sigma}|_m(m') = \widehat{\sigma}(m \uplus m') = \widehat{\sigma}(m) \neq \bot$$

Hence, we have, for all $m' \in \mathsf{Model}_V$, that

$$\mathsf{result}_V(\widehat{\sigma}(m), [\varphi_i : v_i]_{i \in I}|_m, \widehat{\sigma}|_m)(m') = \langle \varnothing \rangle (v_i, \widehat{\sigma}(m))$$

due to the fact that $\mathsf{dom}(\widehat{\sigma}(m)) \setminus \mathsf{dom}(\widehat{\sigma}(m))$. This means that

$$\mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), [\varphi_i : v_i]_{i \in I}|_m, \widehat{\sigma}|_m)) = \{\langle \varnothing \rangle (v_i, \widehat{\sigma}(m))\}$$

Therefore, $\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), \mathtt{fst}\, x, \widehat{\sigma}(m))) = \mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), [\varphi_i : v_i]_{i \in I}|_m, \widehat{\sigma}|_m))$.

(b) Suppose that $\varphi_i(m) = \mathsf{F}$ for all $i \in I$. It follows that $\mathsf{run}(\widehat{\rho}(m), \mathtt{fst}\, x, \widehat{\sigma}(m))(s) = \bot$ for all $s \in \mathbb{B}^{\mathbb{N}}$.

To make this argument precise, assume that $\mathsf{run}(\widehat{\rho}(m), \mathtt{fst}\, x, \widehat{\sigma}(m))(s) \neq \bot$ for some $s \in \mathbb{B}^{\mathbb{N}}$. Then

$$\mathsf{run}(\widehat{\rho}(m), \mathtt{fst}\, x, \widehat{\sigma}(m))(s) = \langle L \rangle (v, \sigma)$$

for some $(v, \sigma) \in \mathsf{Value} \times \mathsf{Store}$. This implies that there exists a derivation in the concrete big-step operational semantics whose conclusion is

$$\widehat{\rho}(m), s \vdash (\mathtt{fst}\, x, \widehat{\sigma}(m)) \Downarrow (v, \widehat{\sigma}(m))$$

This proof tree implies that $\widehat{\rho}(m)(x) = (v, w)$ for some $w \in \mathsf{Value}$. However, this is impossible by construction of $\widehat{\rho}(x)$ since $\varphi_i(m) = \mathsf{F}$ for all $i \in I$. We conclude that $\mathsf{run}(\widehat{\rho}(m), \mathtt{fst}\, x, \widehat{\sigma}(m))(s) = \bot$ for all $s \in \mathbb{B}^{\mathbb{N}}$.

This implies that

$$\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), \mathtt{fst}\, x, \widehat{\sigma}(m))) = \{\bot\}$$

Now, observe that

$$[\varphi_i : v_i]_{i \in I}|_m(m') = [\varphi_i : v_i]_{i \in I}(m \uplus m') = \bot$$

for all $m' \in \mathsf{Model}_V$. Hence, we have, for all $m' \in \mathsf{Model}_V$, that

$$\mathsf{result}_V(\widehat{\sigma}(m), [\varphi_i : v_i]_{i \in I}|_m, \widehat{\sigma}|_m)(m') = \bot$$

meaning that

$$\mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), [\varphi_i : v_i]_{i \in I}|_m, \widehat{\sigma}|_m)) = \{\bot\}$$

Therefore, $\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), \mathtt{fst}\, x, \widehat{\sigma}(m))) = \mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), [\varphi_i : v_i]_{i \in I}|_m, \widehat{\sigma}|_m))$.

This completes our proof of (1).

We now turn our attention to (2). It suffices to show that

$$\left(\bigvee_{i \in I} \varphi_i\right)(m \uplus m') = \mathsf{T} \iff [\varphi_i : v_i]_{i \in I}(m \uplus m') \neq \bot \text{ and } \widehat{\sigma}(m \uplus m')$$

for all $m' \in \mathsf{Model}_V$. Take $m' \in \mathsf{Model}_V$.

($\Longrightarrow$) Suppose that $(\bigvee_{i \in I} \varphi_i)\,(m \uplus m') = \mathsf{T}$. Then it must be the case that $\varphi_i(m) = \mathsf{T}$ for some $i \in I$. This implies that

$$[\varphi_i : v_i]_{i \in I}(m \uplus m') = v_i$$
$$\widehat{\sigma}(m \uplus m') = \widehat{\sigma}(m) \neq \bot$$

as desired.

($\Longleftarrow$) Suppose that $[\varphi_i : v_i]_{i \in I}(m \uplus m') \neq \bot$ and $\widehat{\sigma}(m \uplus m')$. By construction, if $[\varphi_i : v_i]_{i \in I}(m \uplus m') \neq \bot$, then $[\varphi_i : v_i]_{i \in I}(m \uplus m') = v_i$ for some $i \in I$. This means that $\varphi_i(m) = \mathsf{T}$ for some $i \in I$. Hence

$$\left(\bigvee_{i \in I} \varphi_i\right)(m \uplus m') = \mathsf{T}$$

as desired.

This proves (2).

(Snd) Consider $e = \mathsf{snd}\, x$. Apply identical reasoning as in the previous case.

(Ref) Consider $e = \mathtt{ref}\, x$. The only abstract big-step judgement whose conclusion matches $e = \mathtt{ref}\, x$ is Ref. Hence, by inversion of our assumption that $\widehat{\rho} \vdash (e, \widehat{\sigma}) \Downarrow\!\Downarrow (\widehat{v}, \widehat{\sigma}', \psi)$, we get

$$\widehat{\rho} \vdash (\mathtt{ref}\, x, \widehat{\sigma}) \Downarrow\!\Downarrow ([\mathsf{T} : \ell], \widehat{\sigma}[\ell \mapsto \widehat{\rho}(x)], \mathsf{T})$$

such that ℓ is the smallest not in $\mathsf{locs}(\widehat{\rho}, \widehat{\sigma})$.

We will begin by showing (1). It suffices to show that the following two functions have the same image

$$\mathsf{run}(\widehat{\rho}(m), \mathtt{ref}\, x, \widehat{\sigma}(m)) : \mathbb{B}^{\mathbb{N}} \to \mathsf{Result}_\bot$$
$$\mathsf{result}_V(\widehat{\sigma}(m), [\mathsf{T} : \ell]|_m, \widehat{\sigma}[\ell \mapsto \widehat{\rho}(x)]|_m) : \mathsf{Model}_V \to \mathsf{Result}_\bot$$

By the assumption $(\widehat{\rho}(m), \mathtt{ref}\, x, \widehat{\sigma}(m)) \in \mathsf{Config}$, we have that $\mathsf{FV}(\mathtt{ref}\, x) \in \mathsf{dom}(\widehat{\rho}(m))$. Thus, we have that $\widehat{\rho}(m)(x) \neq \bot$. Moreover, we know, because ℓ is not in $\mathsf{locs}(\widehat{\rho}, \widehat{\sigma})$, that ℓ is not in $\mathsf{locs}(\widehat{\rho}(m), \widehat{\sigma}(m))$. These facts allow us to prove, for any $s \in \mathbb{B}^{\mathbb{N}}$, that

$$\widehat{\rho}(m), s \vdash (\mathtt{ref}\, x, \widehat{\sigma}(m)) \Downarrow (\ell, \widehat{\sigma}(m)[\ell \mapsto \widehat{\rho}(m)(x)])$$

using the concrete big-step judgement Ref. Now, consider any $\ell' \notin \mathsf{locs}(\widehat{\rho}(m), \widehat{\sigma}(m))$. We can show, for any $s \in \mathbb{B}^{\mathbb{N}}$, that

$$\widehat{\rho}(m), s \vdash (\mathtt{ref}\, x, \widehat{\sigma}(m)) \Downarrow (\ell', \widehat{\sigma}(m)[\ell' \mapsto \widehat{\rho}(m)(x)])$$

By Theorem B.17, we have that

$$\langle\{\ell\}\rangle(\ell, \widehat{\sigma}(m)[\ell \mapsto \widehat{\rho}(m)(x)]) = \langle\{\ell'\}\rangle(\ell', \widehat{\sigma}(m)[\ell' \mapsto \widehat{\rho}(m)(x)])$$

Hence, for any $s \in \mathbb{B}^{\mathbb{N}}$, we get

$$\mathsf{run}(\widehat{\rho}(m), \mathtt{ref}\, x, \widehat{\sigma}(m)) = \langle\{\ell\}\rangle(\ell, \widehat{\sigma}(m)[\ell \mapsto \widehat{\rho}(m)(x)])$$

due to the fact that $\mathsf{dom}(\widehat{\sigma}(m)[\ell \mapsto \widehat{\rho}(m)(x)]) \setminus \mathsf{dom}(\widehat{\sigma}(m)) = \{\ell\}$. Therefore,

$$\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), \mathtt{ref}\, x, \widehat{\sigma}(m))) = \{\langle\{\ell\}\rangle(\ell, \widehat{\sigma}(m)[\ell \mapsto \widehat{\rho}(m)(x)])\}$$

Now, observe that, for all $m' \in \mathsf{Model}_V$, we have

$$[\mathsf{T} : \ell]|_m(m') = [\mathsf{T} : \ell](m \uplus m') = \ell$$
$$\widehat{\sigma}[\ell \mapsto \widehat{\rho}(x)]|_m(m') = \widehat{\sigma}[\ell \mapsto \widehat{\rho}(x)](m \uplus m') = \widehat{\sigma}(m)[\ell \mapsto \widehat{\rho}(x)(m)] \neq \bot$$

This means, for all $m' \in \mathsf{Model}_V$, that

$$\mathsf{result}_V(\widehat{\sigma}(m), [\mathsf{T} : \ell]|_m, \widehat{\sigma}[\ell \mapsto \widehat{\rho}(x)]|_m)(m') = \langle\{\ell\}\rangle(\ell, \widehat{\sigma}(m)[\ell \mapsto \widehat{\rho}(x)(m)])$$

due to the fact that $\mathsf{dom}(\widehat{\sigma}(m)[\ell \mapsto \widehat{\rho}(x)(m)]) \setminus \mathsf{dom}(\widehat{\sigma}(m)) = \{\ell\}$. Thus,

$$\mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), [\mathsf{T} : \ell]|_m, \widehat{\sigma}[\ell \mapsto \widehat{\rho}(x)]|_m)) = \{\langle\{\ell\}\rangle(\ell, \widehat{\sigma}(m)[\ell \mapsto \widehat{\rho}(x)(m)])\}$$

Hence, $\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), \mathtt{ref}\, x, \widehat{\sigma}(m))) = \mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), [\mathsf{T} : \ell]|_m, \widehat{\sigma}[\ell \mapsto \widehat{\rho}(x)]|_m))$. This completes our proof of (1).

We now turn our attention to (2). It suffices to show that

$$\mathsf{T}(m \uplus m') = \mathsf{T} \iff [\mathsf{T} : \ell](m \uplus m') \neq \bot \text{ and } \widehat{\sigma}[\ell \mapsto \widehat{\rho}(x)](m \uplus m') \neq \bot$$

for all $m' \in \mathsf{Model}_V$. Take $m' \in \mathsf{Model}_V$. Immediately notice that $\mathsf{T}(m \uplus m') = \mathsf{T}$ for all $m' \in \mathsf{Model}_V$. We showed above that $[\mathsf{T} : \ell](m \uplus m') \neq \bot$ and $\widehat{\sigma}[\ell \mapsto \widehat{\rho}(x)](m \uplus m') \neq \bot$ for all $m' \in \mathsf{Model}_V$. Hence, in all cases, we have that $\mathsf{T}(m \uplus m') = \mathsf{T}$, that $[\mathsf{T} : \ell](m \uplus m') \neq \bot$, and that $\widehat{\sigma}[\ell \mapsto \widehat{\rho}(x)](m \uplus m') \neq \bot$. This proves (2).

(Get) Consider $e = !x$. The only abstract big-step judgement whose conclusion matches $e = !x$ is Get. Hence, by inversion of our assumption that $\widehat{\rho} \vdash (e, \widehat{\sigma}) \Downarrow\!\Downarrow (\widehat{v}, \widehat{\sigma}', \psi)$, we get

$$\widehat{\rho} \vdash (!x, \widehat{\sigma}) \Downarrow\!\Downarrow ([\varphi_i : \widehat{\sigma}(\ell_i)]_{i \in I}, \widehat{\sigma}, \bigvee_{i \in I} \varphi_i)$$

such that $\widehat{\rho}(x) = [\varphi_i : \ell_i]_{i \in I} \uplus_{\mathsf{Loc}} \widehat{v}$.

We will begin by showing (1). It suffices to show that the following two functions have the same image

$$\mathsf{run}(\widehat{\rho}(m), !x, \widehat{\sigma}(m)) : \mathbb{B}^{\mathbb{N}} \to \mathsf{Result}_\bot$$
$$\mathsf{result}_V(\widehat{\sigma}(m), [\varphi_i : \widehat{\sigma}(\ell_i)]_{i \in I}|_m, \widehat{\sigma}|_m) : \mathsf{Model}_V \to \mathsf{Result}_\bot$$

By the assumption that $(\widehat{\rho}(m), !x, \widehat{\sigma}(m)) \in \mathsf{Config}$, we know that $\mathsf{locs}(\widehat{\rho}(m)) \cup \mathsf{locs}(\widehat{\sigma}(m)) \subseteq \mathsf{dom}(\widehat{\sigma}(m))$, that $\mathsf{FV}(\rho(x)) = \varnothing$ for all $x \in \mathsf{dom}(\rho)$, and that $\mathsf{FV}(!x) \subseteq \mathsf{dom}(\widehat{\rho}(m))$. There are now two cases:

(a) Suppose that $\varphi_i(m) = \mathsf{T}$ for some $i \in I$. Then we have

$$\widehat{\rho}(m)(x) = ([\varphi_i : \ell_i]_{i \in I} \uplus_{\mathsf{Loc}} \widehat{v})(m) = \ell_i$$

Because $\mathsf{locs}(\widehat{\rho}(m)(x)) \subseteq \mathsf{dom}(\widehat{\sigma}(m))$, we know that $\ell_i \in \mathsf{dom}(\widehat{\sigma}(m))$. Moreover, because $\mathsf{FV}(!x) \subseteq \mathsf{dom}(\widehat{\rho}(m))$, we know that $\widehat{\rho}(m)(x) \neq \bot$. Hence, we can prove, for all $s \in \mathbb{B}^{\mathbb{N}}$, that

$$\widehat{\rho}(m), s \vdash (!x, \widehat{\sigma}(m)) \Downarrow (\widehat{\sigma}(m)(\widehat{\rho}(m)(x)), \widehat{\sigma}(m))$$

using the concrete big-step judgement Get. This implies, for all $s \in \mathbb{B}^{\mathbb{N}}$, that

$$\begin{aligned}\mathsf{run}(\widehat{\rho}(m), !x, \widehat{\sigma}(m))(s) &= \langle\varnothing\rangle(\widehat{\sigma}(m)(\widehat{\rho}(m)(x)), \widehat{\sigma}(m)) \\ &= \langle\varnothing\rangle(\widehat{\sigma}(m)(\ell_i), \widehat{\sigma}(m))\end{aligned}$$

due to the fact that $\mathsf{dom}(\widehat{\sigma}(m)) \setminus \mathsf{dom}(\widehat{\sigma}(m)) = \varnothing$. Hence,

$$\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), !x, \widehat{\sigma}(m))) = \{\langle\varnothing\rangle(\widehat{\sigma}(m)(\ell_i), \widehat{\sigma}(m))\}$$

Observe now, for all $m' \in \mathsf{Model}_V$, that

$$\begin{aligned}[\varphi_i : \widehat{\sigma}(\ell_i)]_{i \in I}|_m(m') = [\varphi_i : \widehat{\sigma}(\ell_i)]_{i \in I}(m \uplus m') = \widehat{\sigma}(\ell_i)(m \uplus m') = \widehat{\sigma}(m)(\ell_i) \neq \bot \\ \widehat{\sigma}|_m(m') = \widehat{\sigma}(m \uplus m') = \widehat{\sigma}(m) \neq \bot\end{aligned}$$

Therefore, we have, for all $m' \in \mathsf{Model}_V$, that

$$\mathsf{result}_V(\widehat{\sigma}(m), [\varphi_i : \widehat{\sigma}(\ell_i)]_{i \in I}|_m, \widehat{\sigma}|_m)(m') = \langle\varnothing\rangle(\widehat{\sigma}(m)(\ell_i), \widehat{\sigma}(m))$$

due to the fact that $\mathsf{dom}(\widehat{\sigma}(m)) \setminus \mathsf{dom}(\widehat{\sigma}(m)) = \varnothing$, meaning that

$$\mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), [\varphi_i : \widehat{\sigma}(\ell_i)]_{i \in I}|_m, \widehat{\sigma}|_m)) = \{\langle\varnothing\rangle(\widehat{\sigma}(m)(\ell_i), \widehat{\sigma}(m))\}$$

Therefore, $\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), !x, \widehat{\sigma}(m))) = \mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), [\varphi_i : \widehat{\sigma}(\ell_i)]_{i \in I}|_m, \widehat{\sigma}|_m))$.

(b) Suppose that $\varphi_i(m) = \mathsf{F}$ for all $i \in I$. It follows that $\mathsf{run}(\widehat{\rho}(m), !x, \widehat{\sigma}(m))(s) = \bot$ for all $s \in \mathbb{B}^{\mathbb{N}}$. To make this argument explicit, assume for the purposes of contradiction that we have $\mathsf{run}(\widehat{\rho}(m), !x, \widehat{\sigma}(m))(s) \neq \bot$ for some $s \in \mathbb{B}^{\mathbb{N}}$. Then

$$\mathsf{run}(\widehat{\rho}(m), !x, \widehat{\sigma}(m))(s) = \langle L\rangle(v, \sigma)$$

for some $(v, \sigma) \in \mathsf{Value} \times \mathsf{Store}$. By definition of run, there must then exist a derivation in the concrete big-step operational semantics whose conclusion is

$$\widehat{\rho}(m), s \vdash (!x, \widehat{\sigma}(m)) \Downarrow (v, \sigma)$$

The existence of this proof tree implies that $\widehat{\rho}(m)(x) \in \mathsf{dom}(\widehat{\sigma}(m))$. In other words, $\widehat{\rho}(m)(x) \in \mathsf{Loc}$. This is impossible by the construction of $\widehat{\rho}(x)$ as $\varphi_i(m) = \mathsf{F}$ for all $i \in I$. We conclude that $\mathsf{run}(\widehat{\rho}(m), !x, \widehat{\sigma}(m))(s) = \bot$ for all $s \in \mathbb{B}^{\mathbb{N}}$.
This implies that

$$\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), !x, \widehat{\sigma}(m))) = \{\bot\}$$

Observe also that, for all $m' \in \mathsf{Model}_V$, we have

$$[\varphi_i : \widehat{\sigma}(\ell_i)]_{i \in I}|_m(m') = [\varphi_i : \widehat{\sigma}(\ell_i)]_{i \in I}(m \uplus m') = \bot$$

meaning that

$$\mathsf{result}_V(\widehat{\sigma}(m), [\varphi_i : \widehat{\sigma}(\ell_i)]_{i \in I}|_m, \widehat{\sigma}|_m)(m') = \bot$$

This implies that

$$\mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), [\varphi_i : \widehat{\sigma}(\ell_i)]_{i \in I}|_m, \widehat{\sigma}|_m)) = \{\bot\}$$

Hence, $\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), !x, \widehat{\sigma}(m))) = \mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), [\varphi_i : \widehat{\sigma}(\ell_i)]_{i\in I}|_m, \widehat{\sigma}|_m))$.
This completes the proof for (1).
We now turn our attention to (2). It suffices to show that

$$\left(\bigvee_{i\in I} \varphi_i\right)(m \uplus m') = \mathsf{T} \iff [\varphi_i : \widehat{\sigma}(\ell_i)]_{i\in I}(m \uplus m') \neq \bot \text{ and } \widehat{\sigma}(m \uplus m') \neq \bot$$

for all $m' \in \mathsf{Model}_V$. Take $m' \in \mathsf{Model}_V$.
($\Longrightarrow$) Suppose that $(\bigvee_{i\in I} \varphi_i)(m \uplus m') = \mathsf{T}$. Then it must be the case that $\varphi_i(m) = \mathsf{T}$ for some $i \in I$. It immediately follows that

$$\begin{aligned} \widehat{\rho}(m)(x) &= ([\varphi_i : \ell_i]_{i\in I} \uplus_{\mathsf{Loc}} \widehat{v})(m) = \ell_i \\ [\varphi_i : \widehat{\sigma}(\ell_i)]_{i\in I}(m \uplus m') &= \widehat{\sigma}(\ell_i)(m \uplus m') = \widehat{\sigma}(m)(\ell_i) \neq \bot \\ \widehat{\sigma}(m \uplus m') &= \widehat{\sigma}(m) \neq \bot \end{aligned}$$

We know that $\widehat{\sigma}(m)(\ell_i) \neq \bot$ due to the fact that $\mathsf{locs}(\widehat{\rho}(m)) \subseteq \mathsf{dom}(\widehat{\sigma}(m))$ and $\widehat{\rho}(m)(x) = \ell_i$.
This completes the claim.
($\Longleftarrow$) Suppose that $[\varphi_i : \widehat{\sigma}(\ell_i)]_{i\in I}(m \uplus m') \neq \bot$ and $\widehat{\sigma}(m \uplus m') \neq \bot$. By construction, the fact that $[\varphi_i : \widehat{\sigma}(\ell_i)]_{i\in I}(m \uplus m') \neq \bot$ implies that $[\varphi_i : \widehat{\sigma}(\ell_i)]_{i\in I}(m \uplus m') = \widehat{\sigma}(m)(\ell_i)$ for some $i \in I$. This implies that $\varphi_i(m) = \mathsf{T}$ for some $i \in I$. Hence,

$$\left(\bigvee_{i\in I} \varphi_i\right)(m \uplus m') = \mathsf{T}$$

as desired.
This completes the proof of (2).

(Set) Consider $e = x \coloneqq y$. The only abstract big-step judgement whose conclusion matches $e = x \coloneqq y$ is Set. Hence, by inversion of our assumption that $\widehat{\rho} \vdash (e, \widehat{\sigma}) \Downarrow (\widehat{v}, \widehat{\sigma}', \psi)$, we get

$$\widehat{\rho} \vdash (x \coloneqq y, \widehat{\sigma}) \Downarrow ([(\textstyle\bigcup_{i\in I} \varphi_i) : ()], [\varphi_i : \widehat{\sigma}[\ell_i \mapsto \widehat{\rho}(y)]]_{i\in I}, \bigvee_{i\in I} \varphi_i)$$

such that $\widehat{\rho}(x) = [\varphi_i : \ell_i]_{i\in I} \uplus_{\mathsf{Loc}} \widehat{v}$.
We will begin by showing (1). It suffices to show that the following two functions have the same image

$$\begin{aligned} \mathsf{run}(\widehat{\rho}(m), x \coloneqq y, \widehat{\sigma}(m)) &: \mathbb{B}^{\mathbb{N}} \to \mathsf{Result}_\bot \\ \mathsf{result}_V\left(\widehat{\sigma}(m), [(\textstyle\bigcup_i \varphi_i) : ()]\,|_m, \left[\varphi_i : \widehat{\sigma}[\ell_i \mapsto \widehat{\rho}(y)]\right]_{i\in I}|_m\right) &: \mathsf{Model}_V \to \mathsf{Result}_\bot \end{aligned}$$

By the assumption that $(\widehat{\rho}(m), !x, \widehat{\sigma}(m)) \in \mathsf{Config}$, we know that $\mathsf{locs}(\widehat{\rho}(m)) \cup \mathsf{locs}(\widehat{\sigma}(m)) \subseteq \mathsf{dom}(\widehat{\sigma}(m))$, that $\mathsf{FV}(\rho(x)) = \varnothing$ for all $x \in \mathsf{dom}(\rho)$, and that $\mathsf{FV}(x \coloneqq y) \subseteq \mathsf{dom}(\widehat{\rho}(m))$. Thus, we have both $\widehat{\rho}(m)(x) \neq \bot$ and $\widehat{\rho}(m)(y) \neq \bot$. There are now two cases:

(a) Suppose that $\varphi_i(m) = \mathsf{T}$ for some $i \in I$. Then we have

$$\widehat{\rho}(m)(x) = ([\varphi_i : \ell_i]_{i\in I} \uplus_{\mathsf{Loc}} \widehat{v}_1)(m) = \ell_i$$

Because $\mathsf{locs}(\widehat{\rho}(m)) \subseteq \mathsf{dom}(\widehat{\sigma}(m))$, we know that $\widehat{\rho}(m)(x) = \ell_i \in \mathsf{dom}(\widehat{\sigma}(m))$. Hence, we can show that, for all $s \in \mathbb{B}^{\mathbb{N}}$,

$$\widehat{\rho}(m), s \vdash (x \coloneqq y, \widehat{\sigma}(m)) \Downarrow ((), \widehat{\sigma}(m)[\widehat{\rho}(m)(x) \mapsto \widehat{\rho}(m)(y)])$$

using the concrete big-step judgement Set. We then get, for all $s \in \mathbb{B}^{\mathbb{N}}$, that

$$\begin{aligned}\mathsf{run}(\widehat{\rho}(m), x \coloneqq y, \widehat{\sigma}(m))(s) &= \langle\varnothing\rangle((), \widehat{\sigma}(m)[\widehat{\rho}(m)(x) \mapsto \widehat{\rho}(m)(y)]) \\ &= \langle\varnothing\rangle((), \widehat{\sigma}(m)[\ell_i \mapsto \widehat{\rho}(m)(y)])\end{aligned}$$

following from the fact that

$$\mathsf{dom}(\widehat{\sigma}(m)[\ell_i \mapsto \widehat{\rho}(m)(y)]) \setminus \mathsf{dom}(\widehat{\sigma}(m)) = \varnothing$$

because $\ell_i \in \mathsf{dom}(\widehat{\sigma}(m))$. Therefore, we have

$$\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), x \coloneqq y, \widehat{\sigma}(m))) = \{\langle\varnothing\rangle((), \widehat{\sigma}(m)[\ell_i \mapsto \widehat{\rho}(m)(y)])\}$$

We also have that, for all $m' \in \mathsf{Model}_V$,

$$\begin{aligned}[(\bigcup_i \varphi_i) : ()]\,|_m(m') &= [(\bigcup_i \varphi_i) : ()]\,(m \uplus m') = () \\ \left[\varphi_i : \widehat{\sigma}[\ell_i \mapsto \widehat{\rho}(y)]\right]_{i\in I}|_m(m') &= \left[\varphi_i : \widehat{\sigma}[\ell_i \mapsto \widehat{\rho}(y)]\right]_{i\in I}(m \uplus m') \\ &= \widehat{\sigma}[\ell_i \mapsto \widehat{\rho}(y)](m \uplus m') \\ &= \widehat{\sigma}(m)[\ell_i \mapsto \widehat{\rho}(m)(y)]\end{aligned}$$

Hence, for all $m' \in \mathsf{Model}_V$, we have

$$\mathsf{result}_V\left(\widehat{\sigma}(m), [(\bigcup_i \varphi_i) : ()]\,|_m, \left[\varphi_i : \widehat{\sigma}[\ell_i \mapsto \widehat{\rho}(y)]\right]_{i\in I}|_m\right)(m') = \langle\varnothing\rangle((), \widehat{\sigma}(m)[\ell_i \mapsto \widehat{\rho}(m)(y)])$$

due to the fact that $\mathsf{dom}(\widehat{\sigma}(m)[\ell_i \mapsto \widehat{\rho}(m)(y)]) \setminus \mathsf{dom}(\widehat{\sigma}(m)) = \varnothing$. Hence,

$$\mathsf{img}\left(\mathsf{result}_V\left(\widehat{\sigma}(m), [(\bigcup_i \varphi_i) : ()]\,|_m, \left[\varphi_i : \widehat{\sigma}[\ell_i \mapsto \widehat{\rho}(y)]\right]_{i\in I}|_m\right)\right) = \{\langle\varnothing\rangle((), \widehat{\sigma}(m)[\ell_i \mapsto \widehat{\rho}(m)(y)])\}$$

meaning that

$$\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), x \coloneqq y, \widehat{\sigma}(m))) = \mathsf{img}\left(\mathsf{result}_V\left(\widehat{\sigma}(m), [(\bigcup_i \varphi_i) : ()]\,|_m, \left[\varphi_i : \widehat{\sigma}[\ell_i \mapsto \widehat{\rho}(y)]\right]_{i\in I}|_m\right)\right)$$

(b) Suppose that $\varphi_i(m) = \mathsf{F}$ for all $i \in I$. It follows that $\mathsf{run}(\widehat{\rho}(m), x \coloneqq y, \widehat{\sigma}(m))(s) = \bot$ for all $s \in \mathbb{B}^{\mathbb{N}}$.
To make this argument explicit, assume for the purposes of contradiction that, for some $s \in \mathbb{B}^{\mathbb{N}}$, we have $\mathsf{run}(\widehat{\rho}(m), x \coloneqq y, \widehat{\sigma}(m))(s) \neq \bot$. Then it is the case that

$$\mathsf{run}(\widehat{\rho}(m), x \coloneqq y, \widehat{\sigma}(m))(s) = \langle L\rangle(v, \sigma)$$

for some $(v, \sigma) \in \mathsf{Value} \times \mathsf{Store}$. The definition of run, implies that there exists a derivation in the concrete big-step operational semantics whose conclusion is

$$\widehat{\rho}(m), s \vdash (x \coloneqq y, \widehat{\sigma}(m)) \Downarrow (v, \sigma)$$

The existence of this proof tree implies that $\widehat{\rho}(m)(x) \in \mathsf{dom}(\widehat{\sigma}(m))$. However, this is impossible by the construction of $\widehat{\rho}(x)$ due to the fact that $\varphi_i(m) = \mathsf{F}$ for all $i \in I$. Hence, we conclude that $\mathsf{run}(\widehat{\rho}(m), x \coloneqq y, \widehat{\sigma}(m))(s) = \bot$ for all $s \in \mathbb{B}^{\mathbb{N}}$.
This implies that

$$\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), x \coloneqq y, \widehat{\sigma}(m))) = \{\bot\}$$

Now, notice that, for all $m' \in \mathsf{Model}_V$, we have

$$\begin{aligned}[(\bigcup_i \varphi_i) : ()]\,(m \uplus m') &= \bot \\ \left(\left[\varphi_i : \widehat{\sigma}[\ell_i \mapsto \widehat{\rho}(y)]\right]_{i\in I}|_m\right)(m \uplus m') &= \bot\end{aligned}$$

This implies that, for all $m' \in \mathsf{Model}_V$,

$$\mathsf{result}_V\left(\widehat{\sigma}(m), [(\bigcup_i \varphi_i) : ()]\,|_m, \left[\varphi_i : \widehat{\sigma}[\ell_i \mapsto \widehat{\rho}(y)]\right]_{i\in I}|_m\right)(m') = \bot$$

meaning that

$$\mathsf{img}\left(\mathsf{result}_V\left(\widehat{\sigma}(m), [(\bigcup_i \varphi_i) : ()]\,|_m, \left[\varphi_i : \widehat{\sigma}[\ell_i \mapsto \widehat{\rho}(y)]\right]_{i\in I}|_m\right)\right) = \{\bot\}$$

Therefore, we conclude that

$$\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), x \coloneqq y, \widehat{\sigma}(m))) = \mathsf{img}\left(\mathsf{result}_V\left(\widehat{\sigma}(m), [(\bigcup_i \varphi_i) : ()]\,|_m, \left[\varphi_i : \widehat{\sigma}[\ell_i \mapsto \widehat{\rho}(y)]\right]_{i\in I}|_m\right)\right)$$

This completes the proof of (1).

We now turn our attention to (2). It suffices to show that

$$\left(\bigvee_{i\in I} \varphi_i\right)(m \uplus m') = \mathsf{T} \iff [(\bigcup_i \varphi_i) : ()]\,(m \uplus m') \neq \bot \text{ and } \left[\varphi_i : \widehat{\sigma}[\ell_i \mapsto \widehat{\rho}(y)]\right]_{i\in I}(m \uplus m') \neq \bot$$

for all $m' \in \mathsf{Model}_V$. Take $m' \in \mathsf{Model}_V$.

($\Longrightarrow$) Suppose that $(\bigvee_{i\in I} \varphi_i)\,(m \uplus m') = \mathsf{T}$. It follows that $\varphi_i(m) = \mathsf{T}$ for some $i \in I$. This implies that

$$[(\bigcup_i \varphi_i) : ()]\,(m \uplus m') = ()$$

$$\left[\varphi_i : \widehat{\sigma}[\ell_i \mapsto \widehat{\rho}(y)]\right]_{i\in I}(m \uplus m') = \widehat{\sigma}[\ell_i \mapsto \widehat{\rho}(y)](m \uplus m') = \widehat{\sigma}(m)[\ell_i \mapsto \widehat{\rho}(m)(y)] \neq \bot$$

We know that $\widehat{\sigma}(m)[\ell_i \mapsto \widehat{\rho}(m)(y)] \neq \bot$ due to the fact that $\widehat{\sigma}(m) \neq \bot$ and $\widehat{\rho}(m)(y) \neq \bot$. This shows the ($\Longrightarrow$) direction.

($\Longleftarrow$) Suppose that $[(\bigcup_i \varphi_i) : ()]\,(m \uplus m') \neq \bot$ and $\left[\varphi_i : \widehat{\sigma}[\ell_i \mapsto \widehat{\rho}(y)]\right]_{i\in I}(m \uplus m') \neq \bot$. By construction, the fact that $\left[\varphi_i : \widehat{\sigma}[\ell_i \mapsto \widehat{\rho}(y)]\right]_{i\in I}(m \uplus m') \neq \bot$ implies

$$\left[\varphi_i : \widehat{\sigma}[\ell_i \mapsto \widehat{\rho}(y)]\right]_{i\in I}(m \uplus m') = \widehat{\sigma}(m)[\ell_i \mapsto \widehat{\rho}(m)(y)]$$

for some $i \in I$. This means that $\varphi_i(m) = \mathsf{T}$ for some $i \in I$. Hence,

$$\left(\bigvee_{i\in I} \varphi_i\right)(m \uplus m') = \mathsf{T}$$

as desired.

This completes our proof of (2).

(Sym) Consider $e = \mathtt{sym}$. The only abstract big-step judgement whose conclusion matches $e = \mathtt{sym}$ is Sym. Hence, by inversion of our assumption that $\widehat{\rho} \vdash (e, \widehat{\sigma}) \Downarrow\!\Downarrow (\widehat{v}, \widehat{\sigma}', \psi)$, we get

$$\widehat{\rho} \vdash (\mathtt{sym}, \widehat{\sigma}) \Downarrow\!\Downarrow ([\alpha : \mathtt{true}, \neg\alpha : \mathtt{false}], \widehat{\sigma}, \mathsf{T})$$

such that α is the smallest not in $\mathsf{symvars}(\widehat{\rho}, \widehat{\sigma})$.

We will begin by showing (1). It suffices to show that the following two functions have the same image

$$\mathsf{run}(\widehat{\rho}(m), \mathtt{sym}, \widehat{\sigma}(m)) : \mathbb{B}^{\mathbb{N}} \to \mathsf{Result}_\bot$$

$$\mathsf{result}_V(\widehat{\sigma}(m), [\alpha : \mathtt{true}, \neg\alpha : \mathtt{false}]|_m, \widehat{\sigma}|_m) : \mathsf{Model}_V \to \mathsf{Result}_\bot$$

We will first consider the image of $\mathsf{run}(\widehat{\rho}(m), \mathtt{sym}, \widehat{\sigma}(m))$. Observe that, by the assumption that $(\widehat{\rho}(m), e, \widehat{\sigma}(m)) \in \mathsf{Config}$, we know that $\widehat{\rho}(m) \neq \bot$ and $\widehat{\sigma}(m) \neq \bot$. Now, there are two sub-cases for the evaluation of $\mathsf{run}(\widehat{\rho}(m), \mathtt{sym}, \widehat{\sigma}(m))$:

(a) Consider $b :: s \in \mathbb{B}^{\mathbb{N}}$ with $b = \mathsf{T}$. Because $b = \mathsf{T}$, we can prove that

$$\widehat{\rho}(m), b :: s \vdash (\mathtt{sym}, \widehat{\sigma}(m)) \Downarrow (\mathtt{true}, \widehat{\sigma}(m))$$

using the concrete big-step judgement SymTrue. Then, by definition of run, we have that

$$\mathsf{run}(\widehat{\rho}(m), \mathtt{sym}, \widehat{\sigma}(m))(b :: s) = \langle\varnothing\rangle(\mathtt{true}, \widehat{\sigma}(m))$$

due to the fact that $\mathsf{dom}(\widehat{\sigma}(m)) \setminus \mathsf{dom}(\widehat{\sigma}(m)) = \varnothing$.

(b) Consider $b :: s \in \mathbb{B}^{\mathbb{N}}$ with $b = \mathsf{F}$. Because $b = \mathsf{F}$, we can prove that

$$\widehat{\rho}(m), b :: s \vdash (\mathtt{sym}, \widehat{\sigma}(m)) \Downarrow (\mathtt{false}, \widehat{\sigma}(m))$$

using the concrete big-step judgement SymFalse. Then, by definition of run, we have that

$$\mathsf{run}(\widehat{\rho}(m), \mathtt{sym}, \widehat{\sigma}(m))(b :: s) = \langle\varnothing\rangle(\mathtt{false}, \widehat{\sigma}(m))$$

due to the fact that $\mathsf{dom}(\widehat{\sigma}(m)) \setminus \mathsf{dom}(\widehat{\sigma}(m)) = \varnothing$.

The cases above imply that

$$\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), \mathtt{sym}, \widehat{\sigma}(m))) = \{\langle\varnothing\rangle(\mathtt{true}, \widehat{\sigma}(m)), \langle\varnothing\rangle(\mathtt{false}, \widehat{\sigma}(m))\}$$

Now, observe that $V = \mathsf{symvars}([\alpha : \mathtt{true}, \neg\alpha : \mathtt{false}], \widehat{\sigma}) \setminus \mathsf{symvars}(\widehat{\rho}, \widehat{\sigma}) = \{\alpha\}$. Thus,

$$\mathsf{Model}_V = \{\{\alpha \mapsto \mathsf{T}\}, \{\alpha \mapsto \bot\}\}$$

Note the following four equivalences, recalling that, by construction, $\alpha \notin \mathsf{symvars}(\widehat{\rho}, \widehat{\sigma})$:

$$[\alpha : \mathtt{true}, \neg\alpha : \mathtt{false}]|_m(\{\alpha \mapsto \mathsf{T}\}) = [\alpha : \mathtt{true}, \neg\alpha : \mathtt{false}](m \uplus \{\alpha \mapsto \mathsf{T}\}) = \mathtt{true}$$
$$\widehat{\sigma}|_m(\{\alpha \mapsto \mathsf{T}\}) = \widehat{\sigma}(m \uplus \{\alpha \mapsto \mathsf{T}\}) = \widehat{\sigma}(m)$$
$$[\alpha : \mathtt{true}, \neg\alpha : \mathtt{false}]|_m(\{\alpha \mapsto \mathsf{F}\}) = [\alpha : \mathtt{true}, \neg\alpha : \mathtt{false}](m \uplus \{\alpha \mapsto \mathsf{F}\}) = \mathtt{false}$$
$$\widehat{\sigma}|_m(\{\alpha \mapsto \mathsf{F}\}) = \widehat{\sigma}(m \uplus \{\alpha \mapsto \mathsf{F}\}) = \widehat{\sigma}(m)$$

Hence, we have, by definition, that

$$\mathsf{result}_V(\widehat{\sigma}(m), [\alpha : \mathtt{true}, \neg\alpha : \mathtt{false}]|_m, \widehat{\sigma}|_m)(\{\alpha \mapsto \mathsf{T}\}) = \langle\varnothing\rangle(\mathtt{true}, \widehat{\sigma}(m))$$
$$\mathsf{result}_V(\widehat{\sigma}(m), [\alpha : \mathtt{true}, \neg\alpha : \mathtt{false}]|_m, \widehat{\sigma}|_m)(\{\alpha \mapsto \mathsf{F}\}) = \langle\varnothing\rangle(\mathtt{false}, \widehat{\sigma}(m))$$

due to the fact that $\mathsf{dom}(\widehat{\sigma}(m)) \setminus \mathsf{dom}(\widehat{\sigma}(m)) = \varnothing$. This means

$$\mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), [\alpha : \mathtt{true}, \neg\alpha : \mathtt{false}]|_m, \widehat{\sigma}|_m)) = \{\langle\varnothing\rangle(\mathtt{true}, \widehat{\sigma}(m)), \langle\varnothing\rangle(\mathtt{false}, \widehat{\sigma}(m))\}$$

Therefore, $\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), \mathtt{sym}, \widehat{\sigma}(m))) = \mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), [\alpha : \mathtt{true}, \neg\alpha : \mathtt{false}]|_m, \widehat{\sigma}|_m))$. This completes our proof of (1)

To prove (2), it suffices to show that

$$\mathsf{T}(m \uplus m') = \mathsf{T} \iff [\alpha : \mathtt{true}, \neg\alpha : \mathtt{false}](m \uplus m') \neq \bot \text{ and } \widehat{\sigma}(m \uplus m') \neq \bot$$

for all $m' \in \mathsf{Model}_V$. Take any $m' \in \mathsf{Model}_V$. Notice immediately that $\mathsf{T}(m \uplus m') = \mathsf{T}$ for all $m' \in \mathsf{Model}_V$. We showed above that $[\alpha : \mathtt{true}, \neg\alpha : \mathtt{false}](m \uplus m') \neq \bot$ and $\widehat{\sigma}(m \uplus m') \neq \bot$ for all $m' \in \mathsf{Model}_V$. Hence, in all cases, we have that $\mathsf{T}(m \uplus m') = \mathsf{T}$, that $[\alpha : \mathtt{true}, \neg\alpha : \mathtt{false}](m \uplus m') \neq \bot$, and that $\widehat{\sigma}(m \uplus m') \neq \bot$. This proves (2).

(Fail) Consider $e = \mathtt{fail}$. The only abstract big-step judgement whose conclusion matches $e = \mathtt{sym}$ is Sym. Hence, by inversion of our assumption that $\widehat{\rho} \vdash (e, \widehat{\sigma}) \Downarrow (\widehat{v}, \widehat{\sigma}', \psi)$, we get

$$\widehat{\rho} \vdash (\mathtt{fail}, \widehat{\sigma}) \Downarrow (\varnothing, \varnothing, \mathsf{F})$$

We will begin by showing (1). It suffices to show that the following two functions have the same image

$$\mathsf{run}(\widehat{\rho}(m), \mathtt{fail}, \widehat{\sigma}(m)) : \mathbb{B}^{\mathbb{N}} \to \mathsf{Result}_\bot$$
$$\mathsf{result}_V(\widehat{\sigma}(m), \varnothing|_m, \varnothing|_m) : \mathsf{Model}_V \to \mathsf{Result}_\bot$$

Notice, there does not exist a concrete big-step judgement that matches $e = \mathtt{fail}$. Hence, by definition of run, we have that

$$\mathsf{run}(\widehat{\rho}(m), \mathtt{fail}, \widehat{\sigma}(m))(s) = \bot$$

for all $s \in \mathbb{B}^{\mathbb{N}}$. Thus,

$$\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), \mathtt{fail}, \widehat{\sigma}(m))) = \{\bot\}$$

Now, observe that $V = \mathsf{symvars}(\varnothing, \widehat{\sigma}) \setminus \mathsf{symvars}(\widehat{\rho}, \widehat{\sigma}) = \varnothing$. Thus, $\mathsf{Model}_V = \{\varnothing\}$. Notice, because there does not exist a φ in the empty symbolic union $\varnothing$ such that $\varphi(m) = \mathsf{T}$, we have $\varnothing|_m(\varnothing) = \bot$. Therefore,

$$\mathsf{result}_V(\widehat{\sigma}(m), \varnothing|_m, \varnothing|_m)(\varnothing) = \bot$$

This means that

$$\mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), \varnothing|_m, \varnothing|_m)) = \{\bot\}$$

Therefore, $\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), \mathtt{fail}, \widehat{\sigma}(m))) = \mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), \varnothing|_m, \varnothing|_m))$. This completes our proof of (1).

Now, we turn our attention to (2). It suffices to show

$$\mathsf{F}(m \uplus m') = \mathsf{T} \iff \varnothing(m \uplus m') \neq \bot \text{ and } \varnothing(m \uplus m') \neq \bot$$

for all $m' \in \mathsf{Model}_V$. Take $m' \in \mathsf{Model}_V$. Notice that $\mathsf{F}(m \uplus m') = \mathsf{F}$ for all $m' \in \mathsf{Model}_V$. We showed above that the symbolic value $\varnothing(m \uplus m') = \bot$ and the symbolic store $\varnothing(m \uplus m') = \bot$ for all $m' \in \mathsf{Model}_V$. Hence, in all cases, we have that $\mathsf{F}(m \uplus m') = \mathsf{F}$, that $\varnothing(m \uplus m') = \bot$, and that the symbolic store $\varnothing(m \uplus m') = \bot$. This proves (2).

(If) Consider $e = \mathtt{if}\ x\ e_1\ e_2$. The only abstract big-step judgement whose conclusion matches $e = \mathtt{if}\ x\ e_1\ e_2$ is If. Hence, by inversion of our assumption that $\widehat{\rho} \vdash (e, \widehat{\sigma}) \Downarrow (\widehat{v}, \widehat{\sigma}', \psi)$, we get

$$\widehat{\rho} \vdash (\mathtt{if}\ x\ e_1\ e_2, \widehat{\sigma}) \Downarrow ([\varphi_1 : \widehat{v}_1, \varphi_2 : \widehat{v}_2], [\varphi_1 : \widehat{\sigma}_1, \varphi_2 : \widehat{\sigma}_2], (\varphi_1 \wedge \psi_1) \vee (\varphi_2 \wedge \psi_2))$$

such that

$$\begin{aligned}
&\widehat{\rho}(x) = [\varphi_1 : \mathtt{true}, \varphi_2 : \mathtt{false}] \uplus_{\mathsf{Bool}} \widehat{v} \\
&\widehat{\rho} \vdash (e_1, \widehat{\sigma}) \Downarrow (\widehat{v}_1, \widehat{\sigma}_1, \psi_1) \\
&\widehat{\rho} \vdash (e_2, \widehat{\sigma}) \Downarrow (\widehat{v}_2, \widehat{\sigma}_2, \psi_2)
\end{aligned}$$

Suppose that $P(\widehat{\rho}, e_1, \widehat{\sigma}, \widehat{v}_1, \widehat{\sigma}_1, \psi_1)$ and $P(\widehat{\rho}, e_2, \widehat{\sigma}, \widehat{v}_2, \widehat{\sigma}_2, \psi_2)$ hold. By $P(\widehat{\rho}, e_1, \widehat{\sigma}, \widehat{v}_1, \widehat{\sigma}_1, \psi_1)$, we have, for all $m_1 \in \mathsf{Model}_{\mathsf{symvars}(\widehat{\rho}, \widehat{\sigma})}$ with $(\widehat{\rho}(m_1), e_1, \widehat{\sigma}(m_1)) \in \mathsf{Config}$, the following two functions have the same image

$$\begin{aligned}
\mathsf{run}(\widehat{\rho}(m_1), e_1, \widehat{\sigma}(m_1)) &: \mathbb{B}^{\mathbb{N}} \to \mathsf{Result}_\bot \\
\mathsf{result}_{V_1}(\widehat{\sigma}(m_1), \widehat{v}_1|_{m_1}, \widehat{\sigma}_1|_{m_1}) &: \mathsf{Model}_{V_1} \to \mathsf{Result}_\bot
\end{aligned}$$

and

$$\psi_1(m_1 \uplus m_1') = \mathsf{T} \iff \widehat{v}_1(m_1 \uplus m_1') \neq \bot \text{ and } \widehat{\sigma}_1(m_1 \uplus m_1') \neq \bot$$

for all $m_1' \in \mathsf{Model}_{V_1}$ where $V_1 = \mathsf{symvars}(\widehat{v}_1, \widehat{\sigma}_1) \setminus \mathsf{symvars}(\widehat{\rho}, \widehat{\sigma})$. By $P(\widehat{\rho}, e_2, \widehat{\sigma}, \widehat{v}_2, \widehat{\sigma}_2, \psi_2)$, we have, for all $m_2 \in \mathsf{Model}_{\mathsf{symvars}(\widehat{\rho}, \widehat{\sigma})}$ with $(\widehat{\rho}(m_2), e_2, \widehat{\sigma}(m_2)) \in \mathsf{Config}$, the following two functions have the same image

$$\begin{aligned}
\mathsf{run}(\widehat{\rho}(m_2), e_2, \widehat{\sigma}(m_2)) &: \mathbb{B}^{\mathbb{N}} \to \mathsf{Result}_\bot \\
\mathsf{result}_{V_1}(\widehat{\sigma}(m_2), \widehat{v}_2|_{m_2}, \widehat{\sigma}_2|_{m_2}) &: \mathsf{Model}_{V_2} \to \mathsf{Result}_\bot
\end{aligned}$$

and

$$\psi_2(m_2 \uplus m_2') = \mathsf{T} \iff \widehat{v}_2(m_2 \uplus m_2') \neq \bot \text{ and } \widehat{\sigma}_2(m_2 \uplus m_2') \neq \bot$$

for all $m_2' \in \mathsf{Model}_{V_2}$ where $V_2 = \mathsf{symvars}(\widehat{v}_2, \widehat{\sigma}_2) \setminus \mathsf{symvars}(\widehat{\rho}, \widehat{\sigma})$.

We will begin by showing (1). It suffices to show that the following two functions have the same image

$$\mathsf{run}(\widehat{\rho}(m), \texttt{if } x\ e_1\ e_2, \widehat{\sigma}(m)) : \mathbb{B}^{\mathbb{N}} \to \mathsf{Result}_\perp$$
$$\mathsf{result}_V(\widehat{\sigma}(m), [\varphi_1 : \widehat{v}_1, \varphi_2 : \widehat{v}_2]|_m, [\varphi_1 : \widehat{\sigma}_1, \varphi_2 : \widehat{\sigma}_2]|_m) : \mathsf{Model}_V \to \mathsf{Result}_\perp$$

We have three sub-cases:

(a) Suppose that $\widehat{\rho}(m)(x) = \texttt{true}$. We will first consider the image of the function

$$\mathsf{run}(\widehat{\rho}(m), \texttt{if } x\ e_1\ e_2, \widehat{\sigma}(m))$$

By the assumption that $(\widehat{\rho}(m), e, \widehat{\sigma}(m)) \in \mathsf{Config}$, we know that $\widehat{\rho}(m) \neq \perp$ and $\widehat{\sigma}(m) \neq \perp$. Moreover, because $m \in \mathsf{Model}_{\mathsf{symvars}(\widehat{\rho},\widehat{\sigma})}$ by construction, notice that $\mathsf{run}(\widehat{\rho}(m), e_1, \widehat{\sigma}(m))$ is a well-defined function. Now, there are two sub-cases:

(i) Consider $s \in \mathbb{B}^{\mathbb{N}}$ with $\mathsf{run}(\widehat{\rho}(m), e_1, \widehat{\sigma}(m))(s) \neq \perp$. Then, by definition,

$$\mathsf{run}(\widehat{\rho}(m), e_1, \widehat{\sigma}(m))(s) = \langle \mathsf{dom}(\sigma') \setminus \mathsf{dom}(\widehat{\sigma}(m)) \rangle (v, \sigma')$$

for some $\langle \mathsf{dom}(\sigma') \setminus \mathsf{dom}(\widehat{\sigma}(m)) \rangle (v, \sigma') \in \mathsf{Result}$. Again, by definition of run, we must have that

$$\widehat{\rho}(m), s \vdash (e_1, \widehat{\sigma}(m)) \Downarrow (v, \sigma')$$

This, along with the fact that $\widehat{\rho}(m)(x) = \texttt{true}$ (meaning $x \in \mathsf{dom}(\widehat{\rho}(m))$), allows us to prove

$$\widehat{\rho}(m), s \vdash (\texttt{if } x\ e_1\ e_2, \widehat{\sigma}(m)) \Downarrow (v, \sigma')$$

using the concrete big-step judgement IfTrue. By the definition of run, this implies that

$$\mathsf{run}(\widehat{\rho}(m), \texttt{if } x\ e_1\ e_2, \widehat{\sigma}(m))(s) = \langle \mathsf{dom}(\sigma') \setminus \mathsf{dom}(\widehat{\sigma}(m)) \rangle (v, \sigma')$$

meaning that $\mathsf{run}(\widehat{\rho}(m), \texttt{if } x\ e_1\ e_2, \widehat{\sigma}(m))(s) = \mathsf{run}(\widehat{\rho}(m), e_1, \widehat{\sigma}(m))(s)$.

(ii) Consider $s \in \mathbb{B}^{\mathbb{N}}$ with $\mathsf{run}(\widehat{\rho}(m), e_1, \widehat{\sigma}(m))(s) = \perp$. Then there does not exist a bitstream $s \in \mathbb{B}^{\mathbb{N}}$ such that $\widehat{\rho}(m), s \vdash (\texttt{if } x\ e_1\ e_2, \widehat{\sigma}(m)) \Downarrow (v, \sigma')$ for some $(v, \sigma') \in \mathsf{Value} \times \mathsf{Store}$.
To make this claim explicit, assume, for the purposes of contradiction, that there exists $s \in \mathbb{B}^{\mathbb{N}}$ such that $\widehat{\rho}(m), s \vdash (\texttt{if } x\ e_1\ e_2, \widehat{\sigma}(m)) \Downarrow (v, \sigma')$ for some $(v, \sigma') \in \mathsf{Value} \times \mathsf{Store}$. There are two concrete big-step judgements that match this conclusion: IfTrue and IfFalse. Notice, because $\widehat{\rho}(m)(x) = \texttt{true}$, we could not have used IfFalse as the final step of our derivation, meaning we must have used IfTrue. Therefore, by inversion of the proof tree corresponding to $\widehat{\rho}(m), s \vdash (\texttt{if } x\ e_1\ e_2, \widehat{\sigma}(m)) \Downarrow (v, \sigma')$, we get that

$$\widehat{\rho}(m), s \vdash (e_1, \widehat{\sigma}(m)) \Downarrow (v, \sigma')$$

This means, by definition of run, that

$$\mathsf{run}(\widehat{\rho}(m), e_1, \widehat{\sigma}(m))(s) = \langle \mathsf{dom}(\sigma') \setminus \mathsf{dom}(\widehat{\sigma}(m)) \rangle (v, \sigma')$$

which is impossible. Hence, we conclude that there does not exist $s \in \mathbb{B}^{\mathbb{N}}$ such that $\widehat{\rho}(m), s \vdash (\texttt{if } x\ e_1\ e_2, \widehat{\sigma}(m)) \Downarrow (v, \sigma')$ for some $(v, \sigma') \in \mathsf{Value} \times \mathsf{Store}$.
This implies, by the definition of run, that, for all $s \in \mathbb{B}^{\mathbb{N}}$,

$$\mathsf{run}(\widehat{\rho}(m), \texttt{if } x\ e_1\ e_2, \widehat{\sigma}(m))(s) = \perp$$

meaning that $\mathsf{run}(\widehat{\rho}(m), \texttt{if } x\ e_1\ e_2, \widehat{\sigma}(m))(s) = \mathsf{run}(\widehat{\rho}(m), e_1, \widehat{\sigma}(m))(s)$.

The cases above imply, for all $s \in \mathbb{B}^{\mathbb{N}}$, that

$$\mathsf{run}(\widehat{\rho}(m), \texttt{if } x\ e_1\ e_2, \widehat{\sigma}(m))(s) = \mathsf{run}(\widehat{\rho}(m), e_1, \widehat{\sigma}(m))(s)$$

This implies that

$$\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), \texttt{if } x\ e_1\ e_2, \widehat{\sigma}(m))) = \mathsf{img}(\mathsf{run}(\widehat{\rho}(m), e_1, \widehat{\sigma}(m)))$$

Now, notice that $m \in \mathsf{Model}_{\mathsf{symvars}(\widehat{\rho},\widehat{\sigma})}$ by construction. Moreover, because we have $(\widehat{\rho}(m), \texttt{if}\ x\ e_1\ e_2, \widehat{\sigma}(m)$ Config, we know that $\mathsf{locs}(\widehat{\rho}(m)) \cup \mathsf{locs}(\widehat{\sigma}(m)) \subseteq \mathsf{dom}(\widehat{\sigma}(m))$, $\mathsf{FV}(\widehat{\rho}(m)(x)) = \varnothing$ for all $x \in \mathsf{dom}(\widehat{\rho}(m))$, and $\mathsf{FV}(\texttt{if}\ x\ e_1\ e_2) \subseteq \mathsf{dom}(\widehat{\rho}(m))$. The final condition implies that $\mathsf{FV}(e_1) \subseteq \widehat{\rho}(m)$. Thus, we know that $(\widehat{\rho}(m), e_1, \widehat{\sigma}(m)) \in \mathsf{Config}$. Therefore,

$$\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), e_1, \widehat{\sigma}(m))) = \mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), \widehat{v}_1|_m, \widehat{\sigma}_1|_m))$$

by the inductive hypothesis. Finally, note that because $\widehat{\rho}(m)(x) = \texttt{true}$ implies that $\varphi_1(m) = \mathsf{T}$, we have

$$[\varphi_1 : \widehat{v}_1, \varphi_2 : \widehat{v}_2]|_m(m') = [\varphi_1 : \widehat{v}_1, \varphi_2 : \widehat{v}_2]|(m \uplus m') = \widehat{v}_1(m \uplus m') = \widehat{v}_1|_m(m')$$
$$[\varphi_1 : \widehat{\sigma}_1, \varphi_2 : \widehat{\sigma}_2]|_m(m') = [\varphi_1 : \widehat{\sigma}_1, \varphi_2 : \widehat{\sigma}_2]|(m \uplus m') = \widehat{\sigma}_1(m \uplus m') = \widehat{\sigma}_1|_m(m')$$

for all $m' \in \mathsf{Model}_{(\mathsf{symvars}(\widehat{v}_1,\widehat{\sigma}_1) \cup \mathsf{symvars}(\widehat{v}_2,\widehat{\sigma}_2)) \setminus \mathsf{symvars}(\widehat{\rho},\widehat{\sigma})}$. In other words, we have that

$$[\varphi_1 : \widehat{v}_1, \varphi_2 : \widehat{v}_2]|_m = \widehat{v}_1|_m$$
$$[\varphi_1 : \widehat{\sigma}_1, \varphi_2 : \widehat{\sigma}_2]|_m = \widehat{\sigma}_1|_m$$

Putting everything together we have that

$$\begin{aligned}\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), \texttt{if}\ x\ e_1\ e_2, \widehat{\sigma}(m))) &= \mathsf{img}(\mathsf{run}(\widehat{\rho}(m), e_1, \widehat{\sigma}(m))) \\ &= \mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), \widehat{v}_1|_m, \widehat{\sigma}_1|_m)) \\ &= \mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), [\varphi_1 : \widehat{v}_1, \varphi_2 : \widehat{v}_2]|_m, [\varphi_1 : \widehat{\sigma}_1, \varphi_2 : \widehat{\sigma}_2]|_m))\end{aligned}$$

as desired.

(b) Suppose that $\widehat{\rho}(m)(x) = \texttt{false}$. Apply identical reasoning as in the case above.

(c) Suppose that $\widehat{\rho}(m)(x) \neq \texttt{true}$ and $\widehat{\rho}(m)(x) \neq \texttt{false}$. Then there does not exist a bitstream $s \in \mathbb{B}^{\mathbb{N}}$ such that $\widehat{\rho}(m), s \vdash (\texttt{if}\ x\ e_1\ e_2, \widehat{\sigma}(m)) \Downarrow (v, \sigma')$ for some $(v, \sigma') \in \mathsf{Value} \times \mathsf{Store}$.
To make this make explicit, assume, for the purposes of contradiction, that there exists $s \in \mathbb{B}^{\mathbb{N}}$ such that $\widehat{\rho}(m), s \vdash (\texttt{if}\ x\ e_1\ e_2, \widehat{\sigma}(m)) \Downarrow (v, \sigma')$ for some $(v, \sigma') \in \mathsf{Value} \times \mathsf{Store}$. There are two concrete big-step judgements that match this conclusion: IfTrue and IfFalse. If we used IfTrue as the final step of our derivation, we get that $\widehat{\rho}(m)(x) = \texttt{true}$ by inversion of the proof tree. This is impossible since $\widehat{\rho}(m)(x) \neq \texttt{true}$. Likewise, if we used IfFalse as the final step of our derivation, we get that $\widehat{\rho}(m)(x) = \texttt{false}$ by inversion of the proof tree. This too is impossible since $\widehat{\rho}(m)(x) \neq \texttt{false}$. Thus, our assumption was false, and we conclude that there does not exist a bitstream $s \in \mathbb{B}^{\mathbb{N}}$ such that $\widehat{\rho}(m), s \vdash (\texttt{if}\ x\ e_1\ e_2, \widehat{\sigma}(m)) \Downarrow (v, \sigma')$ for some $(v, \sigma') \in \mathsf{Value} \times \mathsf{Store}$.
This implies, for all $s \in \mathbb{B}^{\mathbb{N}}$, that

$$\mathsf{run}(\widehat{\rho}(m), \texttt{if}\ x\ e_1\ e_2, \widehat{\sigma}(m))(s) = \bot$$

meaning that

$$\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), \texttt{if}\ x\ e_1\ e_2, \widehat{\sigma}(m))) = \{\bot\}$$

Now, notice that $\widehat{\rho}(m)(x) \neq \texttt{true}$ and $\widehat{\rho}(m)(x) \neq \texttt{false}$ implies that $\varphi_1(m) = \mathsf{F}$ and $\varphi_2(m) = \mathsf{F}$. Hence, we have

$$[\varphi_1 : \widehat{v}_1, \varphi_2 : \widehat{v}_2]|_m(m') = [\varphi_1 : \widehat{v}_1, \varphi_2 : \widehat{v}_2]|(m \uplus m') = \bot$$
$$[\varphi_1 : \widehat{\sigma}_1, \varphi_2 : \widehat{\sigma}_2]|_m(m') = [\varphi_1 : \widehat{\sigma}_1, \varphi_2 : \widehat{\sigma}_2]|(m \uplus m') = \bot$$

for all $m' \in (\mathsf{symvars}(\widehat{v}_1, \widehat{\sigma}_1) \cup \mathsf{symvars}(\widehat{v}_2, \widehat{\sigma}_2)) \setminus \mathsf{symvars}(\widehat{\rho}, \widehat{\sigma})$. Therefore, by definition, we have

$$\mathsf{result}_V(\widehat{\sigma}(m), [\varphi_1 : \widehat{v}_1, \varphi_2 : \widehat{v}_2]|_m, [\varphi_1 : \widehat{\sigma}_1, \varphi_2 : \widehat{\sigma}_2]|_m)(m') = \bot$$

for all $m' \in (\mathsf{symvars}(\widehat{v}_1, \widehat{\sigma}_1) \cup \mathsf{symvars}(\widehat{v}_2, \widehat{\sigma}_2)) \setminus \mathsf{symvars}(\widehat{\rho}, \widehat{\sigma})$, meaning that

$$\mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), [\varphi_1 : \widehat{v}_1, \varphi_2 : \widehat{v}_2]|_m, [\varphi_1 : \widehat{\sigma}_1, \varphi_2 : \widehat{\sigma}_2]|_m)) = \{\bot\}$$

So,

$$\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), \texttt{if } x\ e_1\ e_2, \widehat{\sigma}(m))) = \mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), [\varphi_1 : \widehat{v}_1, \varphi_2 : \widehat{v}_2]|_m, [\varphi_1 : \widehat{\sigma}_1, \varphi_2 : \widehat{\sigma}_2]|_m))$$

Therefore, in all cases, we have that

$$\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), \texttt{if } x\ e_1\ e_2, \widehat{\sigma}(m))) = \mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), [\varphi_1 : \widehat{v}_1, \varphi_2 : \widehat{v}_2]|_m, [\varphi_1 : \widehat{\sigma}_1, \varphi_2 : \widehat{\sigma}_2]|_m))$$

so we are done. This completes our proof of (1).

Now, we turn our attention to (2). It suffices to show

$$\big((\varphi_1 \vee \psi_1) \wedge (\varphi_2 \vee \psi_2)\big)(m \uplus m') = \mathsf{T} \iff [\varphi_1 : \widehat{v}_1, \varphi_2 : \widehat{v}_2](m \uplus m') \neq \bot, [\varphi_1 : \widehat{\sigma}_1, \varphi_2 : \widehat{\sigma}_2](m \uplus m') \neq \bot$$

for all $m' \in \mathsf{Model}_V$. Take $m' \in \mathsf{Model}_V$.

($\Longrightarrow$) Suppose that $\big((\varphi_1 \vee \psi_1) \wedge (\varphi_2 \vee \psi_2)\big)(m \uplus m') = \mathsf{T}$. Then we must have that either $(\varphi_1 \wedge \psi_1)(m \uplus m') = \mathsf{T}$ or that $(\varphi_2 \wedge \psi_2)(m \uplus m') = \mathsf{T}$.

(a) Suppose $(\varphi_1 \wedge \psi_1)(m \uplus m') = \mathsf{T}$. Write $m' = m_1 \uplus m''$ for $m_1 \in \mathsf{Model}_{V_1}$ and $m'' \in \mathsf{Model}_{V \setminus V_1}$. By the inductive hypothesis, we know that

$$\psi_1(m \uplus m_1) = \mathsf{T} \iff \widehat{v}_1(m \uplus m_1) \neq \bot \text{ and } \widehat{\sigma}_1(m \uplus m_1) \neq \bot$$

Our assumption implies that $\psi_1(m \uplus m_1) = \mathsf{T}$, so it must be the case that

$$[\varphi_1 : \widehat{v}_1, \varphi_2 : \widehat{v}_2](m \uplus m') = \widehat{v}_1(m \uplus m_1) \neq \bot$$
$$[\varphi_1 : \widehat{\sigma}_1, \varphi_2 : \widehat{\sigma}_2](m \uplus m') = \widehat{\sigma}_1(m \uplus m_1) \neq \bot$$

as desired.

(b) Suppose $(\varphi_2 \wedge \psi_2)(m \uplus m') = \mathsf{T}$. Following the logic from the previous case, we know that $[\varphi_1 : \widehat{v}_1, \varphi_2 : \widehat{v}_2](m \uplus m') \neq \bot$ and $[\varphi_1 : \widehat{\sigma}_1, \varphi_2 : \widehat{\sigma}_2](m \uplus m') \neq \bot$.

This proves the ($\Longrightarrow$) direction.

($\Longleftarrow$) Suppose that $[\varphi_1 : \widehat{v}_1, \varphi_2 : \widehat{v}_2](m \uplus m') \neq \bot$ and $[\varphi_1 : \widehat{\sigma}_1, \varphi_2 : \widehat{\sigma}_2](m \uplus m') \neq \bot$. By construction, this can only be the case if either (i) $\varphi_1(m \uplus m') = \mathsf{T}$, $\widehat{v}_1(m \uplus m_1) \neq \bot$, and $\widehat{\sigma}_1(m \uplus m_1) \neq \bot$ where $m' = m_1 \uplus m''$ for $m_1 \in \mathsf{Model}_{V_1}$ and $m'' \in \mathsf{Model}_{V \setminus V_1}$ or (ii) $\varphi_2(m \uplus m') = \mathsf{T}$, $\widehat{v}_2(m \uplus m_2) \neq \bot$, and $\widehat{\sigma}_2(m \uplus m_2) \neq \bot$ where $m' = m_2 \uplus m'''$ for $m_2 \in \mathsf{Model}_{V_2}$ and $m''' \in \mathsf{Model}_{V \setminus V_2}$.

(a) Suppose that $\varphi_1(m \uplus m') = \mathsf{T}$, $\widehat{v}_1(m \uplus m_1) \neq \bot$, and $\widehat{\sigma}_1(m \uplus m_1) \neq \bot$. By the inductive hypothesis, we know that

$$\psi_1(m \uplus m_1) = \mathsf{T} \iff \widehat{v}_1(m \uplus m_1) \neq \bot \text{ and } \widehat{\sigma}_1(m \uplus m_1) \neq \bot$$

Hence, it must be the case that $\psi_1(m \uplus m_1) = \mathsf{T}$. Therefore, we have $(\varphi_1 \wedge \psi_1)(m \uplus m') = \mathsf{T}$, meaning that

$$\big((\varphi_1 \vee \psi_1) \wedge (\varphi_2 \vee \psi_2)\big)(m \uplus m') = \mathsf{T}$$

as desired.

(b) Suppose that $\varphi_2(m \uplus m') = \mathsf{T}$, $\widehat{v}_2(m \uplus m_2) \neq \bot$, and $\widehat{\sigma}_2(m \uplus m_2) \neq \bot$. Following the logic from the previous case, we have that $\big((\varphi_1 \vee \psi_1) \wedge (\varphi_2 \vee \psi_2)\big)(m \uplus m') = \mathsf{T}$, as desired.

This shows the ($\Longleftarrow$) direction, and completes the proof of (2).

(App) Consider $e = x\ y$. The only abstract big-step judgement whose conclusion matches $e = x\ y$ is App. Hence, by inversion of our assumption that $\widehat{\rho} \vdash (e, \widehat{\sigma}) \Downarrow\!\Downarrow (\widehat{v}, \widehat{\sigma}', \psi)$, we get

$$\widehat{\rho} \vdash (x\ y, \widehat{\sigma}) \Downarrow\!\Downarrow ([\varphi_i : \widehat{v}_i]_{i \in I}, [\varphi_i : \widehat{\sigma}_i]_{i \in I}, \bigvee_{i \in I} (\varphi_i \wedge \psi_i))$$

such that

$$\widehat{\rho}(x) = [\varphi_i : \texttt{clo}(\lambda x_i.e_i, \widehat{\rho}_i)]_{i\in I} \uplus_{\text{Closure}} \widehat{v}$$
$$\widehat{\rho}_i[x_i \mapsto \widehat{\rho}(y)] \vdash (e_i, \widehat{\sigma}) \Downarrow (\widehat{v}_i, \widehat{\sigma}_i, \psi_i) \qquad \text{for all } i \in I$$

Suppose that $P(\widehat{\rho}_i[x_i \mapsto \widehat{\rho}(y)], e_i, \widehat{\sigma}, \widehat{v}_i, \widehat{\sigma}_i, \psi_i)$ holds for all $i \in I$. Then, for each $i \in I$, we have that, for all $m_i \in \mathsf{Model}_{\mathsf{symvars}(\widehat{\rho}_i[x_i\mapsto\widehat{\rho}(y)],\widehat{\sigma})}$ with $(\widehat{\rho}_i[x \mapsto \rho(y)](m_i), e_i, \widehat{\sigma}(m_i)) \in \mathsf{Config}$, the following two functions have the same image

$$\mathsf{run}(\widehat{\rho}_i[x_i \mapsto \widehat{\rho}(y)](m_i), e_i, \widehat{\sigma}(m_i)) : \mathbb{B}^{\mathbb{N}} \to \mathsf{Result}_\perp$$
$$\mathsf{result}_{V_i}(\widehat{\sigma}(m_i), \widehat{v}_i|_{m_i}, \widehat{\sigma}_i|_{m_i}) : \mathsf{Model}_{V_i} \to \mathsf{Result}_\perp$$

and

$$\psi_i(m_i \uplus m_i') = \mathsf{T} \iff \widehat{v}_i(m_i \uplus m_i') \neq \perp \text{ and } \widehat{\sigma}_i(m_i \uplus m_i') \neq \perp$$

for all $m_i' \in \mathsf{Model}_{V_i}$ where $V_i = \mathsf{symvars}(\widehat{v}_i, \widehat{\sigma}_i) \setminus \mathsf{symvars}(\widehat{\rho}_i[x_i \mapsto \widehat{\rho}(y)], \widehat{\sigma})$.
We will begin by showing (1). It suffices to show that the following two functions have the same image

$$\mathsf{run}(\widehat{\rho}(m), x\ y, \widehat{\sigma}(m)) : \mathbb{B}^{\mathbb{N}} \to \mathsf{Result}_\perp$$
$$\mathsf{result}_V(\widehat{\sigma}(m), [\varphi_i : \widehat{v}_i]_{i\in I}|_m, [\varphi_i : \widehat{\sigma}_i]_{i\in I}|_m) : \mathsf{Model}_V \to \mathsf{Result}_\perp$$

By the assumption that $(\widehat{\rho}(m), x\ y, \widehat{\sigma}(m)) \in \mathsf{Config}$, we know $\mathsf{locs}(\widehat{\rho}(m)) \cup \mathsf{locs}(\widehat{\sigma}(m)) \subseteq \mathsf{dom}(\widehat{\sigma}(m))$, $\mathsf{FV}(\widehat{\rho}(m)(x)) = \varnothing$ for all $x \in \mathsf{dom}(\widehat{\rho}(m))$, and $\mathsf{FV}(x\ y) \subseteq \mathsf{dom}(\widehat{\rho}(m))$. Therefore, by assumption, we have that $x, y \in \mathsf{dom}(\widehat{\rho}(m))$, meaning that $\widehat{\rho}(m)(x) \neq \perp$ and $\widehat{\rho}(m)(y) \neq \perp$. Now, we split into two cases:

(a) Suppose that $\varphi_i(m) = \mathsf{T}$ for some $i \in I$. By definition

$$\widehat{\rho}(m)(x) = ([\varphi_i : \texttt{clo}(\lambda x_i.e_i, \widehat{\rho}_i)]_{i\in I} \uplus_{\text{Closure}} \widehat{v})(m) = \texttt{clo}(\lambda x_i.e_i, \widehat{\rho}_i(m))$$

noting that, because we know that $\widehat{\rho}(m)(x) \neq \perp$, it must also be the case that $\widehat{\rho}_i(m) \neq \perp$ by the definition of abstract closures.
We would now like to show that $(\widehat{\rho}_i(m)[x_i \mapsto \widehat{\rho}(m)(y)], e_i, \widehat{\sigma}(m)) \in \mathsf{Config}$. Observe, because $\mathsf{locs}(\widehat{\rho}(m)) \cup \mathsf{locs}(\widehat{\sigma}(m)) \subseteq \mathsf{dom}(\widehat{\sigma}(m))$, we know $\mathsf{locs}(\texttt{clo}(\lambda x_i.e_i, \widehat{\rho}_i(m))) \subseteq \mathsf{locs}(\widehat{\rho}(m)) \subseteq \mathsf{dom}(\widehat{\sigma}(m))$. This implies $\mathsf{locs}(\widehat{\rho}_i(m)) \subseteq \mathsf{dom}(\widehat{\sigma}(m))$. Moreover, we know that $\mathsf{locs}(\widehat{\rho}(m)(y)) \subseteq \mathsf{locs}(\widehat{\rho}(m)) \subseteq \mathsf{dom}(\widehat{\sigma}(m))$. Combining these facts, we can conclude that $\mathsf{locs}(\widehat{\rho}_i(m)[x_i \mapsto \widehat{\rho}(m)(y)]) \subseteq \mathsf{dom}(\widehat{\sigma}(m))$, meaning that

$$\mathsf{locs}(\widehat{\rho}_i(m)[x_i \mapsto \widehat{\rho}(m)(y)]) \cup \mathsf{locs}(\widehat{\sigma}(m)) \subseteq \mathsf{dom}(\widehat{\sigma}(m))$$

Notice, due to the fact that $\mathsf{FV}(\widehat{\rho}(m)(x)) = \varnothing$ for all $x \in \mathsf{dom}(\widehat{\rho}(m))$, we know that

$$\mathsf{FV}(\widehat{\rho}_i(m)(x)) = \varnothing \text{ for all } x \in \mathsf{dom}(\widehat{\rho}_i(m))$$

Moreover, because $\mathsf{FV}(\texttt{clo}(\lambda x_i.e_i, \widehat{\rho}_i(m))) = \varnothing$, we have that $\mathsf{FV}(\lambda x_i.e_i) \subseteq \mathsf{dom}(\widehat{\rho}_i(m))$. This implies that $\mathsf{FV}(e_i) \subseteq \mathsf{dom}(\widehat{\rho}_i(m)) \cup \{x_i\}$. Thus, we can conclude $\mathsf{FV}(e_i) \subseteq \mathsf{dom}(\widehat{\rho}_i(m)[x_i \mapsto \widehat{\rho}(m)(y)])$. Altogether, these facts imply that

$$(\widehat{\rho}_i(m)[x_i \mapsto \widehat{\rho}(m)(y)], e_i, \widehat{\sigma}(m)) \in \mathsf{Config}$$

With this important context, we will now show that

$$\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), x\ y, \widehat{\sigma}(m))) = \mathsf{img}(\mathsf{run}(\widehat{\rho}_i(m)[x_i \mapsto \widehat{\rho}(m)(y)], e_i, \widehat{\sigma}(m)))$$

There are two sub-cases:

(i) Consider $s \in \mathbb{B}^{\mathbb{N}}$ such that $\mathsf{run}(\widehat{\rho}_i(m)[x_i \mapsto \widehat{\rho}(m)(y)], e_i, \widehat{\sigma}(m)) \neq \bot$. Then, by definition

$$\mathsf{run}(\widehat{\rho}_i(m)[x_i \mapsto \widehat{\rho}(m)(y)], e_i, \widehat{\sigma}(m))(s) = \langle \mathsf{dom}(\sigma) \setminus \mathsf{dom}(\widehat{\sigma}(m)) \rangle (v, \sigma)$$

for some $(v, \sigma) \in \mathsf{Value} \times \mathsf{Store}$. This implies that there exists a derivation in the concrete big-step operational semantics whose conclusion is

$$\widehat{\rho}_i(m)[x_i \mapsto \widehat{\rho}(m)(y)], s \vdash (e_i, \widehat{\sigma}(m)) \Downarrow (v, \sigma)$$

Because $\widehat{\rho}(m)(x) = \mathtt{clo}(\lambda x_i.e_i, \widehat{\rho}_i(m))$, we can prove that

$$\widehat{\rho}(m), s \vdash (x\ y, \widehat{\sigma}(m)) \Downarrow (v, \sigma)$$

using the concrete big-step judgement App. Therefore, by definition of run, we have that

$$\mathsf{run}(\widehat{\rho}(m), x\ y, \widehat{\sigma}(m))(s) = \langle \mathsf{dom}(\sigma) \setminus \mathsf{dom}(\widehat{\sigma}(m)) \rangle (v, \sigma)$$

meaning that $\mathsf{run}(\widehat{\rho}(m), x\ y, \widehat{\sigma}(m))(s) = \mathsf{run}(\widehat{\rho}_i(m)[x_i \mapsto \widehat{\rho}(m)(y)], e_i, \widehat{\sigma}(m))(s)$.

(ii) Consider $s \in \mathbb{B}^{\mathbb{N}}$ such that $\mathsf{run}(\widehat{\rho}_i(m)[x_i \mapsto \widehat{\rho}(m)(y)], e_i, \widehat{\sigma}(m)) = \bot$. Then it must be the case that $\mathsf{run}(\widehat{\rho}(m), x\ y, \widehat{\sigma}(m)) = \bot$.
To make this claim explicit, assume, for the purposes of contradiction, that there exists $s \in \mathbb{B}^{\mathbb{N}}$ such that

$$\mathsf{run}(\widehat{\rho}(m), x\ y, \widehat{\sigma}(m))(s) = \langle \mathsf{dom}(\sigma) \setminus \mathsf{dom}(\widehat{\sigma}(m)) \rangle (v, \sigma)$$

for some $(v, \sigma) \in \mathsf{Value} \times \mathsf{Store}$. Then there exists a derivation in the concrete big-step operational semantics whose conclusion is

$$\widehat{\rho}(m), s \vdash (x\ y, \widehat{\sigma}(m)) \vdash (v, \sigma)$$

Recall, we know that $\widehat{\rho}(m)(x) = \mathtt{clo}(\lambda x_i.e_i, \widehat{\rho}_i(m))$. Thus, by inversion of the proof tree for $\widehat{\rho}(m), s \vdash (x\ y, \widehat{\sigma}(m)) \vdash (v, \sigma)$, we conclude that

$$\widehat{\rho}_i(m)[x_i \mapsto \widehat{\rho}(m)(y)], s \vdash (e_i, \widehat{\sigma}(m)) \Downarrow (v, \sigma)$$

This implies

$$\mathsf{run}(\widehat{\rho}_i(m)[x_i \mapsto \widehat{\rho}(m)(y)], e_i, \widehat{\sigma}(m))(s) = \langle \mathsf{dom}(\sigma) \setminus \mathsf{dom}(\widehat{\sigma}(m)) \rangle (v, \sigma)$$

which is impossible. Therefore, we have that

$$\mathsf{run}(\widehat{\rho}(m), x\ y, \widehat{\sigma}(m))(s) = \bot = \mathsf{run}(\widehat{\rho}_i(m)[x_i \mapsto \widehat{\rho}(m)(y)], e_i, \widehat{\sigma}(m))(s)$$

The cases above demonstrate that

$$\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), x\ y, \widehat{\sigma}(m))) = \mathsf{img}(\mathsf{run}(\widehat{\rho}_i(m)[x_i \mapsto \widehat{\rho}(m)(y)], e_i, \widehat{\sigma}(m)))$$

Because $m \in \mathsf{Model}_{\mathsf{symvars}(\widehat{\rho}_i[x_i \mapsto \widehat{\rho}(y)](m), \widehat{\sigma}(m))}$, and $(\widehat{\rho}_i[x_i \mapsto \widehat{\rho}(y)](m), e_i, \widehat{\sigma}(m)) \in \mathsf{Config}$, we also know that

$$\mathsf{img}(\mathsf{run}(\widehat{\rho}_i(m)[x_i \mapsto \widehat{\rho}(m)(y)], e_i, \widehat{\sigma}(m))) = \mathsf{img}(\mathsf{result}_{V_i}(\widehat{\sigma}(m), \widehat{v}_i|_m, \widehat{\sigma}_i|_m))$$

by the inductive hypothesis. Finally, because $\varphi_i(m) = \mathsf{T}$, notice that

$$[\varphi_i : \widehat{v}_i]_{i \in I}|_m(m') = [\varphi_i : \widehat{v}_i]_{i \in I}(m \uplus m') = \widehat{v}_i(m \uplus m') = \widehat{v}_i|_m(m')$$
$$[\varphi_i : \widehat{\sigma}_i]_{i \in I}|_m(m') = [\varphi_i : \widehat{\sigma}_i]_{i \in I}(m \uplus m') = \widehat{\sigma}_i(m \uplus m') = \widehat{\sigma}_i|_m(m')$$

meaning

$$\mathsf{result}_V(\widehat{\sigma}(m), [\varphi_i : \widehat{v}_i]_{i \in I}|_m, [\varphi_i : \widehat{\sigma}_i]_{i \in I}|_m)(m') = \mathsf{result}_{V_i}(\widehat{\sigma}(m), \widehat{v}_i|_m, \widehat{\sigma}_i|_m)(m')$$

for all $m' \in \mathsf{Model}_{\mathsf{symvars}([\varphi_i : \widehat{v}_i]_{i \in I}|_m, [\varphi_i : \widehat{\sigma}_i]_{i \in I}|_m) \setminus \mathsf{symvars}(\widehat{\rho}, \widehat{\sigma})}$. This directly implies that

$$\mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), [\varphi_i : \widehat{v}_i]_{i \in I}|_m, [\varphi_i : \widehat{\sigma}_i]_{i \in I}|_m)) = \mathsf{img}(\mathsf{result}_{V_i}(\widehat{\sigma}(m), \widehat{v}_i|_m, \widehat{\sigma}_i|_m))$$

Therefore, we can conclude

$$\begin{aligned}\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), x\ y, \widehat{\sigma}(m))) &= \mathsf{img}(\mathsf{run}(\widehat{\rho}_i(m)[x_i \mapsto \widehat{\rho}(m)(y)], e_i, \widehat{\sigma}(m))) \\ &= \mathsf{img}(\mathsf{result}_{V_i}(\widehat{\sigma}(m), \widehat{v}_i|_m, \widehat{\sigma}_i|_m)) \\ &= \mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), [\varphi_i : \widehat{v}_i]_{i\in I}|_m, [\varphi_i : \widehat{\sigma}_i]_{i\in I}|_m))\end{aligned}$$

as desired.

(b) Suppose that $\varphi_i(m) = \mathsf{F}$ for all $i \in I$. Then we have that

$$\widehat{\rho}(m)(x) = ([\varphi_i : \mathtt{clo}(\lambda x_i.e_i, \widehat{\rho}_i)]_{i\in I} \uplus_{\mathsf{Closure}} \widehat{v})(m) \notin \mathsf{Closure}$$

Hence, it must be the case that $\mathsf{run}(\widehat{\rho}(m), x\ y, \widehat{\sigma}(m))(s) = \bot$ for all $s \in \mathbb{B}^{\mathbb{N}}$.
To make this claim explicit, assume, for the purposes of contradiction, that

$$\mathsf{run}(\widehat{\rho}(m), x\ y, \widehat{\sigma}(m))(s) = \langle \mathsf{dom}(\sigma) \setminus \mathsf{dom}(\widehat{\sigma}(m))\rangle(v, \sigma)$$

for some $(v, \sigma) \in \mathsf{Value} \times \mathsf{Store}$. This implies that there exists a derivation in the concrete big-step operational semantics whose conclusion is

$$\widehat{\rho}(m), s \vdash (x\ y, \widehat{\sigma}(m)) \Downarrow (v, \sigma)$$

By inversion of this proof tree, we then conclude that $\widehat{\rho}(m)(x) = \mathtt{clo}(\lambda x'.e', \rho')$ for some $\mathtt{clo}(\lambda x'.e', \rho') \in \mathsf{Closure}$, which is impossible.
Therefore, we conclude $\mathsf{run}(\widehat{\rho}(m), x\ y, \widehat{\sigma}(m))(s) = \bot$ for all $s \in \mathbb{B}^{\mathbb{N}}$, meaning that

$$\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), x\ y, \widehat{\sigma}(m))) = \{\bot\}$$

Likewise, notice that, because $\varphi_i(m) = \mathsf{F}$ for all $i \in I$, we have that

$$\begin{aligned}[\varphi_i : \widehat{v}_i]_{i\in I}|_m(m') &= [\varphi_i : \widehat{v}_i]_{i\in I}(m \uplus m') = \bot \\ [\varphi_i : \widehat{\sigma}_i]_{i\in I}|_m(m') &= [\varphi_i : \widehat{\sigma}_i]_{i\in I}(m \uplus m') = \bot\end{aligned}$$

meaning

$$\mathsf{result}_V(\widehat{\sigma}(m), [\varphi_i : \widehat{v}_i]_{i\in I}|_m, [\varphi_i : \widehat{\sigma}_i]_{i\in I}|_m)(m') = \bot$$

for all $m' \in \mathsf{Model}_{\mathsf{symvars}([\varphi_i:\widehat{v}_i]_{i\in I}|_m, [\varphi_i:\widehat{\sigma}_i]_{i\in I}|_m) \setminus \mathsf{symvars}(\widehat{\rho},\widehat{\sigma})}$. This directly implies that

$$\mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), [\varphi_i : \widehat{v}_i]_{i\in I}|_m, [\varphi_i : \widehat{\sigma}_i]_{i\in I}|_m)) = \{\bot\}$$

Therefore, we conclude that

$$\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), x\ y, \widehat{\sigma}(m))) = \mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), [\varphi_i : \widehat{v}_i]_{i\in I}|_m, [\varphi_i : \widehat{\sigma}_i]_{i\in I}|_m))$$

as desired.

This completes our proof of (1).
Now, we turn our attention to (2). It suffices to show

$$\left(\bigvee_{i\in I}(\varphi_i \wedge \psi_i)\right)(m \uplus m') = \mathsf{T} \iff [\varphi_i : \widehat{v}_i]_{i\in I}(m \uplus m') \neq \bot \text{ and } [\varphi_i : \widehat{\sigma}_i]_{i\in I}(m \uplus m') \neq \bot$$

for all $m' \in \mathsf{Model}_V$. Consider any $m' \in \mathsf{Model}_V$.
($\Longrightarrow$) Suppose that $(\bigvee_{i\in I}(\varphi_i \wedge \psi_i))(m \uplus m') = \mathsf{T}$. This implies $(\varphi_i \wedge \psi_i)(m \uplus m') = \mathsf{T}$ and for some $i \in I$. Write $m' = m_i \uplus m''$ for some $m_i \in \mathsf{Model}_{V_i}$. We then have that $\psi_i(m \uplus m_i) = \mathsf{T}$ and $\varphi_i(m \uplus m') = \mathsf{T}$. By the inductive hypothesis, we then get

$$\psi_i(m \uplus m_i) = \mathsf{T} \iff \widehat{v}_i(m \uplus m_i) \neq \bot \text{ and } \widehat{\sigma}_i(m \uplus m_i) \neq \bot$$

Because $\psi_i(m \uplus m_i) = \mathsf{T}$, we have $\widehat{v}_i(m \uplus m_i) \neq \bot$ and $\widehat{\sigma}_i(m \uplus m_i) \neq \bot$. This means

$$[\varphi_i : \widehat{v}_i]_{i \in I}(m \uplus m') = \widehat{v}_i(m \uplus m_i) \neq \bot$$
$$[\varphi_i : \widehat{\sigma}_i]_{i \in I}(m \uplus m') = \widehat{\sigma}_i(m \uplus m_i) \neq \bot$$

as desired.
($\Longleftarrow$) Suppose that $[\varphi_i : \widehat{v}_i]_{i \in I}(m \uplus m') \neq \bot$ and $[\varphi_i : \widehat{\sigma}_i]_{i \in I}(m \uplus m') \neq \bot$. Repeat the argument for ($\Longrightarrow$) in the opposite direction.
This proves (2).

(Let) Consider $e = \texttt{let}\ x = e_1\ \texttt{in}\ e_2$. The only abstract big-step judgement whose conclusion matches $e = \texttt{let}\ x = e_1\ \texttt{in}\ e_2$ is Let. Hence, by inversion of our assumption that $\widehat{\rho} \vdash (e, \widehat{\sigma}) \Downarrow\!\Downarrow (\widehat{v}, \widehat{\sigma}', \psi)$, we get

$$\widehat{\rho} \vdash (\texttt{let}\ x = e_1\ \texttt{in}\ e_2, \widehat{\sigma}) \Downarrow\!\Downarrow ([\psi_1 : \widehat{v}_2], [\psi_1 : \widehat{\sigma}_2], \psi_1 \wedge \psi_2)$$

such that

$$\widehat{\rho} \vdash (e_1, \widehat{\sigma}) \Downarrow\!\Downarrow (\widehat{v}_1, \widehat{\sigma}_1, \psi_1)$$
$$\widehat{\rho}[x \mapsto \widehat{v}_1] \vdash (e_2, \widehat{\sigma}_1) \Downarrow\!\Downarrow (\widehat{v}_2, \widehat{\sigma}_2, \psi_2)$$

Suppose that $P(\widehat{\rho}, e_1, \widehat{\sigma}, \widehat{v}_1, \widehat{\sigma}_1, \psi_1)$ and $P(\widehat{\rho}[x \mapsto \widehat{v}_1], e_2, \widehat{\sigma}_1, \widehat{v}_2, \widehat{\sigma}_2, \psi_2)$ hold. By the assumption $P(\widehat{\rho}, e_1, \widehat{\sigma}, \widehat{v}_1, \widehat{\sigma}_1, \psi_1)$, we have, for all $m_1 \in \mathsf{Model}_{\mathsf{symvars}(\widehat{\rho}, \widehat{\sigma})}$ with $(\widehat{\rho}(m_1), e_1, \widehat{\sigma}(m_1)) \in \mathsf{Config}$, the following two functions have the same image

$$\mathsf{run}(\widehat{\rho}(m_1), e_1, \widehat{\sigma}(m_1)) : \mathbb{B}^{\mathbb{N}} \to \mathsf{Result}_\bot$$
$$\mathsf{result}_{V_1}(\widehat{\sigma}(m_1), \widehat{v}_1|_{m_1}, \widehat{\sigma}_1|_{m_1}) : \mathsf{Model}_{V_1} \to \mathsf{Result}_\bot$$

and

$$\psi_1(m_1 \uplus m_1') = \mathsf{T} \Longleftrightarrow \widehat{v}_1(m_1 \uplus m_1') \neq \bot \text{ and } \widehat{\sigma}_1(m_1 \uplus m_1') \neq \bot$$

for all $m_1' \in \mathsf{Model}_{V_1}$ where $V_1 = \mathsf{symvars}(\widehat{v}_1, \widehat{\sigma}_1) \setminus \mathsf{symvars}(\widehat{\rho}, \widehat{\sigma})$. By the assumption $P(\widehat{\rho}[x \mapsto \widehat{v}_1], e_2, \widehat{\sigma}_1, \widehat{v}_2, \widehat{\sigma}_2, \psi_2)$, we have, for all $m_2 \in \mathsf{Model}_{\mathsf{symvars}(\widehat{\rho}[x \mapsto \widehat{v}_1], \widehat{\sigma}_1)}$ with $(\widehat{\rho}[x \mapsto \widehat{v}_1](m_2), e_2, \widehat{\sigma}_1(m_2)) \in \mathsf{Config}$, the following two functions have the same image

$$\mathsf{run}(\widehat{\rho}[x \mapsto \widehat{v}_1](m_2), e_2, \widehat{\sigma}_1(m_2)) : \mathbb{B}^{\mathbb{N}} \to \mathsf{Result}_\bot$$
$$\mathsf{result}_{V_2}(\widehat{\sigma}_1(m_2), \widehat{v}_2|_{m_2}, \widehat{\sigma}_2|_{m_2}) : \mathsf{Model}_{V_2} \to \mathsf{Result}_\bot$$

and

$$\psi_2(m_2 \uplus m_2') = \mathsf{T} \Longleftrightarrow \widehat{v}_2(m_2 \uplus m_2') \neq \bot \text{ and } \widehat{\sigma}_2(m_2 \uplus m_2') \neq \bot$$

for all $m_2' \in \mathsf{Model}_{V_2}$ where $V_2 = \mathsf{symvars}(\widehat{v}_2, \widehat{\sigma}_2) \setminus \mathsf{symvars}(\widehat{\rho}[x \mapsto \widehat{v}_1], \widehat{\sigma}_1)$.
We will begin by showing (1). It suffices to show that the following two functions have the same image

$$\mathsf{run}(\widehat{\rho}(m), \texttt{let}\ x = e_1\ \texttt{in}\ e_2, \widehat{\sigma}(m)) : \mathbb{B}^{\mathbb{N}} \to \mathsf{Result}_\bot$$
$$\mathsf{result}_V(\widehat{\sigma}(m), [\psi_1 : \widehat{v}_2]|_m, [\psi_1 : \widehat{\sigma}_2]|_m) : \mathsf{Model}_V \to \mathsf{Result}_\bot$$

We will break this proof into two parts:

$$\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), \texttt{let}\ x = e_1\ \texttt{in}\ e_2, \widehat{\sigma}(m))) \subseteq \mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), [\psi_1 : \widehat{v}_2]|_m, [\psi_1 : \widehat{\sigma}_2]|_m))$$
$$\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), \texttt{let}\ x = e_1\ \texttt{in}\ e_2, \widehat{\sigma}(m))) \supseteq \mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), [\psi_1 : \widehat{v}_2]|_m, [\psi_1 : \widehat{\sigma}_2]|_m))$$

Before we begin, recall that $(\widehat{\rho}(m), \texttt{let}\ x = e_1\ \texttt{in}\ e_2, \widehat{\sigma}(m)) \in \mathsf{Config}$ by assumption. Hence, we know that $\mathsf{locs}(\widehat{\rho}(m)) \cup \mathsf{locs}(\widehat{\sigma}(m)) \subseteq \mathsf{dom}(\widehat{\sigma}(m))$, that $\mathsf{FV}(\texttt{let}\ x = e_1\ \texttt{in}\ e_2) \subseteq \mathsf{dom}(\widehat{\rho}(m))$, and $\mathsf{FV}(\widehat{\rho}(m)(x)) \neq \emptyset$ for all $x \in \mathsf{dom}(\widehat{\rho}(m))$.

($\subseteq$) Now, consider any $s \in \mathbb{B}^{\mathbb{N}}$ and write $s \rightsquigarrow s_1, s_2$. We will consider several cases for the evaluation of $\mathsf{run}(\widehat{\rho}(m), \mathtt{let}\ x = e_1\ \mathtt{in}\ e_2, \widehat{\sigma}(m))(s)$ and show that, in each case,

$$\mathsf{run}(\widehat{\rho}(m), \mathtt{let}\ x = e_1\ \mathtt{in}\ e_2, \widehat{\sigma}(m))(s) \in \mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), [\psi_1 : \widehat{v}_2]|_m, [\psi_1 : \widehat{\sigma}_2]|_m))$$

(a) Suppose that $\mathsf{run}(\widehat{\rho}(m), e_1, \widehat{\sigma}(m))(s_1) = \bot$. This fact implies that $\mathsf{run}(\widehat{\rho}(m), \mathtt{let}\ x = e_1\ \mathtt{in}\ e_2, \widehat{\sigma}(m))(s) = \bot$.

To be precise, assume, for the purposes of contradiction, that

$$\mathsf{run}(\widehat{\rho}(m), \mathtt{let}\ x = e_1\ \mathtt{in}\ e_2, \widehat{\sigma}(m))(s) \neq \bot$$

By definition of run, this implies that there exists a derivation in the concrete big-step operational semantics whose conclusion is

$$\widehat{\rho}(m), s \vdash (\mathtt{let}\ x = e_1\ \mathtt{in}\ e_2, \widehat{\sigma}(m)) \Downarrow (v, \sigma)$$

for some $(v, \sigma) \in \mathsf{Value} \times \mathsf{Store}$. Recall, $s \rightsquigarrow s_1, s_2$ by assumption. Hence, by inversion of the proof tree for $\widehat{\rho}(m), s \vdash (\mathtt{let}\ x = e_1\ \mathtt{in}\ e_2, \widehat{\sigma}(m)) \Downarrow (v, \sigma)$, we have that

$$\widehat{\rho}(m), s_1 \vdash (e_1, \widehat{\sigma}(m)) \Downarrow (v_1, \sigma_1)$$
$$\widehat{\rho}(m)[x \mapsto v_1], s_2 \vdash (e_2, \sigma_1) \Downarrow (v, \sigma)$$

This implies, by the definition of run, that

$$\mathsf{run}(\widehat{\rho}(m), e_1, \widehat{\sigma}(m))(s_1) = \langle \mathsf{dom}(\sigma) \setminus \mathsf{dom}(\sigma_1) \rangle (v_1, \sigma_1)$$

which is impossible. Therefore, $\mathsf{run}(\widehat{\rho}(m), \mathtt{let}\ x = e_1\ \mathtt{in}\ e_2, \widehat{\sigma}(m))(s) = \bot$.
Now, we must show $\bot \in \mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), [\psi_1 : \widehat{v}_2]|_m, [\psi_1 : \widehat{\sigma}_2]|_m))$. Notice that we have $(\widehat{\rho}(m), e_1, \widehat{\sigma}(m)) \in \mathsf{Config}$ due to the fact that $\mathsf{FV}(e_1) \subseteq \mathsf{FV}(\mathtt{let}\ x = e_1\ \mathtt{in}\ e_2)$. Hence, by the inductive hypothesis,

$$\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), e_1, \widehat{\sigma}(m))) = \mathsf{img}(\mathsf{result}_{V_1}(\widehat{\sigma}(m), \widehat{v}_1|_m, \widehat{\sigma}_1|_m))$$

Therefore, because $\bot \in \mathsf{img}(\mathsf{run}(\widehat{\rho}(m), e_1, \widehat{\sigma}(m)))$, there must exist $m_1 \in \mathsf{Model}_{V_1}$ such that

$$\mathsf{result}_{V_1}(\widehat{\sigma}(m), \widehat{v}_1|_m, \widehat{\sigma}_1|_m)(m_1) = \bot$$

This means, by the definition of result_{V_1}, that we have at least one of $\widehat{v}_1(m \uplus m_1) = \bot$ or $\widehat{\sigma}_1(m \uplus m_1) = \bot$. By the inductive hypothesis, this occurs if and only if $\psi_1(m \uplus m_1) = \mathsf{F}$. Therefore, for any $m_2 \in \mathsf{Model}_{V_2}$, we get

$$[\psi_1 : \widehat{v}_2]|_m(m_1 \uplus m_2) = [\psi_1 : \widehat{v}_2](m \uplus m_1 \uplus m_2) = \bot$$
$$[\psi_1 : \widehat{\sigma}_2]|_m(m_1 \uplus m_2) = [\psi_1 : \widehat{\sigma}_2](m \uplus m_1 \uplus m_2) = \bot$$

using the fact that $m_1 \uplus m_2 \in \mathsf{Model}_V$ for any $m_1 \in \mathsf{Model}_{V_1}$ and $m_2 \in \mathsf{Model}_{V_2}$. Hence, by the definition of result_V,

$$\mathsf{result}_V(\widehat{\sigma}(m), [\psi_1 : \widehat{v}_2]|_m, [\psi_1 : \widehat{\sigma}_2]|_m)(m_1 \uplus m_2) = \bot$$

for any $m_2 \in \mathsf{Model}_{V_2}$, meaning

$$\bot \in \mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), [\psi_1 : \widehat{v}_2]|_m, [\psi_1 : \widehat{\sigma}_2]|_m))$$

as desired.

(b) Suppose that $\mathsf{run}(\widehat{\rho}(m), e_1, \widehat{\sigma}(m))(s_1) \neq \bot$. Then, by definition of run, we have that

$$\mathsf{run}(\widehat{\rho}(m), e_1, \widehat{\sigma}(m))(s_1) = \langle \mathsf{dom}(\sigma_1) \setminus \mathsf{dom}(\widehat{\sigma}(m)) \rangle (v_1, \sigma_1)$$

for some $(v_1, \sigma_1) \in \mathsf{Value} \times \mathsf{Store}$. Observe, because $(\widehat{\rho}(m), e_1, \widehat{\sigma}(m)) \in \mathsf{Config}$, we have that $\mathsf{FV}(v_1) = \varnothing$, that $\mathsf{dom}(\sigma_1) = \mathsf{dom}(\widehat{\sigma}(m)) \uplus \mathsf{dom}(\sigma'')$ for some $\sigma'' \in \mathsf{Store}$, that $\mathsf{locs}(\sigma_1) \subseteq$

$\mathsf{dom}(\sigma_1)$, and that $v_1 \in \mathsf{Loc}$ implies that $v_1 \in \mathsf{dom}(\sigma_1)$ by Theorem B.14. These facts imply that

$$(\widehat{\rho}(m)[x \mapsto v_1], e_2, \sigma_1) \in \mathsf{Config}$$

due to $\mathsf{FV}(e_2) \subseteq \mathsf{dom}(\widehat{\rho}(m)) \cup \{x\} = \mathsf{dom}(\widehat{\rho}(m)[x \mapsto v_1])$. As in the previous case, the inductive hypothesis implies that

$$\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), e_1, \widehat{\sigma}(m))) = \mathsf{img}(\mathsf{result}_{V_1}(\widehat{\sigma}(m), \widehat{v}_1|_m, \widehat{\sigma}_1|_m))$$

Therefore,

$$\mathsf{result}_{V_1}(\widehat{\sigma}(m), \widehat{v}_1|_m, \widehat{\sigma}_1|_m)(m_1) = \langle \mathsf{dom}(\sigma_1) \setminus \mathsf{dom}(\widehat{\sigma}(m)) \rangle (v_1, \sigma_1)$$

for some $m_1 \in \mathsf{Model}_{V_1}$ such that $v_1 = \widehat{v}_1(m \uplus m_1)$ and $\sigma_1 = \widehat{\sigma}_1(m \uplus m_1)$. By the inductive hypothesis, we then know that $\psi_1(m \uplus m_1) = \mathsf{T}$. Moreover, it implies that

$$(\widehat{\rho}(m)[x \mapsto v_1], e_2, \sigma_1) = (\widehat{\rho}[x \mapsto \widehat{v}_1](m \uplus m_1), e_2, \widehat{\sigma}_1(m \uplus m_1)) \in \mathsf{Config}$$

We must now break into sub-cases for $\mathsf{run}(\widehat{\rho}[x \mapsto \widehat{v}_1](m \uplus m_1), e_2, \widehat{\sigma}_1(m \uplus m_1))(s_2)$:

(i) Suppose that $\mathsf{run}(\widehat{\rho}[x \mapsto \widehat{v}_1](m \uplus m_1), e_2, \widehat{\sigma}_1(m \uplus m_1))(s_2) = \bot$. Then it must be the case that $\mathsf{run}(\widehat{\rho}(m), \texttt{let } x = e_1 \texttt{ in } e_2, \widehat{\sigma}(m))(s) = \bot$.

To be precise, assume, for the purposes of contradiction, that

$$\mathsf{run}(\widehat{\rho}(m), \texttt{let } x = e_1 \texttt{ in } e_2, \widehat{\sigma}(m))(s) \neq \bot$$

By definition of run, this implies that there exists a derivation in the concrete big-step operational semantics whose conclusion is

$$\widehat{\rho}(m), s \vdash (\texttt{let } x = e_1 \texttt{ in } e_2, \widehat{\sigma}(m)) \Downarrow (v, \sigma)$$

Recall, $s \rightsquigarrow s_1, s_2$ Hence, by inversion of the proof tree for $\widehat{\rho}(m), s \vdash (\texttt{let } x = e_1 \texttt{ in } e_2, \widehat{\sigma}(m)) \Downarrow (v, \sigma)$, we have that

$$\widehat{\rho}(m), s_1 \vdash (e_1, \widehat{\sigma}(m)) \Downarrow (v_1', \sigma_1')$$
$$\widehat{\rho}(m)[x \mapsto v_1'], s_2 \vdash (e_2, \sigma_1') \Downarrow (v, \sigma)$$

This implies, by definition of run, that

$$\mathsf{run}(\widehat{\rho}(m), e_1, \widehat{\sigma}(m))(s_1) = \langle \mathsf{dom}(\sigma_1') \setminus \mathsf{dom}(\widehat{\sigma}(m)) \rangle (v_1', \sigma_1')$$
$$\mathsf{run}(\widehat{\rho}(m)[x \mapsto v_1'], e_2, \sigma_1')(s_2) = \langle \mathsf{dom}(\sigma) \setminus \mathsf{dom}(\sigma_1') \rangle (v, \sigma)$$

By Theorem B.17, we know that

$$\langle \mathsf{dom}(\widehat{\sigma}_1(m \uplus m_1)) \setminus \mathsf{dom}(\widehat{\sigma}(m)) \rangle (\widehat{v}_1(m \uplus m_1), \widehat{\sigma}_1(m \uplus m_1)) = \langle \mathsf{dom}(\sigma_1') \setminus \mathsf{dom}(\widehat{\sigma}(m)) \rangle (v_1', \sigma_1')$$

For convenience, say that $\mathsf{dom}(\widehat{\sigma}_1(m \uplus m_1)) \setminus \mathsf{dom}(\widehat{\sigma}(m)) = \{\ell_1, \ldots, \ell_n\}$ and that $\mathsf{dom}(\sigma_1') \setminus \mathsf{dom}(\widehat{\sigma}(m)) = \{\ell_1', \ldots, \ell_n'\}$. By Theorem B.9, there must exist locations $f_1, \ldots, f_n$ disjoint from $\mathsf{locs}(\widehat{v}_1(m \uplus m_1), \widehat{\sigma}_1(m \uplus m_1), v_1', \sigma_1')$ such that

$$(\ell_1\ f_1) \cdots (\ell_n\ f_n) \cdot \widehat{v}_1(m \uplus m_1), \widehat{\sigma}_1(m \uplus m_1) = (\ell_1'\ f_1) \cdots (\ell_n'\ f_n) \cdot (v_1', \sigma_1')$$

Let $\pi = (\ell_n\ f_n) \cdots (\ell_1\ f_1) \cdot (\ell_1'\ f_1) \cdots (\ell_n'\ f_n)$. Observe that

$$\pi \cdot (v_1', \sigma_1') = (\widehat{v}_1(m \uplus m_1), \widehat{\sigma}_1(m \uplus m_1))$$

and that

$$\pi \cdot \widehat{\rho}(m) = \widehat{\rho}(m)$$

because $\ell_1, \ldots, \ell_n, \ell_1', \ldots, \ell_n', f_1, \ldots, f_n \notin \mathsf{dom}(\widehat{\sigma}(m))$ and, by assumption, we know $\mathsf{locs}(\widehat{\rho}(m)) \subseteq \mathsf{dom}(\widehat{\sigma}(m))$. This implies that

$$\pi \cdot \widehat{\rho}[x \mapsto v_1'] = \widehat{\rho}(m)[x \mapsto \widehat{v}_1(m \uplus m_1)] = \widehat{\rho}[x \mapsto \widehat{v}_1](m \uplus m_1)$$

by definition. Therefore, combining that $\widehat{\rho}(m)[x \mapsto v_1'], s_2 \vdash (e_2, \sigma_1') \Downarrow (v, \sigma)$, that $\pi \cdot \widehat{\rho}(m)[x \mapsto v_1'] = \widehat{\rho}[x \mapsto \widehat{v}_1](m \uplus m_1)$, and that $\pi \cdot \sigma_1' = \widehat{\sigma}_1(m \uplus m_1)$, we get that

$$\widehat{\rho}[x \mapsto \widehat{v}_1](m \uplus m_1), s_2 \vdash (e_2, \widehat{\sigma}_1(m \uplus m_1)) \Downarrow (\pi \cdot v, \pi \cdot \sigma)$$

by Theorem B.15. Finally, by definition of run, we get that

$$\mathsf{run}(\widehat{\rho}[x \mapsto \widehat{v}_1](m \uplus m_1), e_2, \widehat{\sigma}_1(m \uplus m_1))(s_2) = \langle \mathsf{dom}(\pi \cdot \sigma) \setminus \mathsf{dom}(\widehat{\sigma}_1(m \uplus m')) \rangle (\pi \cdot v, \pi \cdot \sigma)$$

which contradicts $\mathsf{run}(\widehat{\rho}[x \mapsto \widehat{v}_1](m \uplus m_1), e_2, \widehat{\sigma}_1(m \uplus m_1))(s_2) = \bot$. Hence, we conclude that $\mathsf{run}(\widehat{\rho}(m), \mathtt{let}\ x = e_1\ \mathtt{in}\ e_2, \widehat{\sigma}(m))(s) = \bot$.
We must now show that $\bot \in \mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), [\psi_1 : \widehat{v}_2]|_m, [\psi_1 : \widehat{\sigma}_2]|_m))$. Because $(\widehat{\rho}[x \mapsto \widehat{v}_1](m \uplus m_1), e_2, \widehat{\sigma}_1(m \uplus m_1)) \in \mathsf{Config}$, we know that

$$\mathsf{img}(\mathsf{run}(\widehat{\rho}[x \mapsto \widehat{v}](m \uplus m_1), e_2, \widehat{\sigma}_1(m \uplus m_1))) = \mathsf{img}(\mathsf{result}_{V_2}(\widehat{\sigma}(m \uplus m_1), \widehat{v}_2|_{m \uplus m_1}, \widehat{\sigma}_2|_{m \uplus m_1}))$$

by the inductive hypothesis. Thus, our assumption that $\mathsf{run}(\widehat{\rho}[x \mapsto \widehat{v}_1](m \uplus m_1), e_2, \widehat{\sigma}_1(m \uplus m_1))(s_2) = \bot$ implies

$$\mathsf{result}_{V_2}(\widehat{\sigma}(m \uplus m_1), \widehat{v}_2|_{m \uplus m_1}, \widehat{\sigma}_2|_{m \uplus m_1})(m_2) = \bot$$

for some $m_2 \in \mathsf{Model}_{V_2}$. This implies that at least one of $\widehat{v}_2(m \uplus m_1 \uplus m_2) = \bot$ or $\widehat{\sigma}_2(m \uplus m_1 \uplus m_2) = \bot$. If $\widehat{v}_2(m \uplus m_1 \uplus m_2) = \bot$, then

$$[\psi_1 : \widehat{v}_2]|_m(m_1 \uplus m_2) = [\psi_1 : \widehat{v}_2](m \uplus m_1 \uplus m_2) = \widehat{v}_2(m \uplus m_1 \uplus m_2) = \bot$$

meaning

$$\mathsf{result}_V(\widehat{\sigma}(m), [\psi_1 : \widehat{v}_2]|_m, [\psi_1 : \widehat{\sigma}_2]|_m)(m_1 \uplus m_2) = \bot$$

Likewise, if $\widehat{\sigma}_2(m \uplus m_1 \uplus m_2) = \bot$, then

$$[\psi_1 : \widehat{\sigma}_2]|_m(m_1 \uplus m_2) = [\psi_1 : \widehat{\sigma}_2](m \uplus m_1 \uplus m_2) = \widehat{\sigma}_2(m \uplus m_1 \uplus m_2) = \bot$$

meaning

$$\mathsf{result}_V(\widehat{\sigma}(m), [\psi_1 : \widehat{v}_2]|_m, [\psi_1 : \widehat{\sigma}_2]|_m)(m_1 \uplus m_2) = \bot$$

In both cases, $\bot \in \mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), [\psi_1 : \widehat{v}_2]|_m, [\psi_1 : \widehat{\sigma}_2]|_m))$.

(ii) Suppose that $\mathsf{run}(\widehat{\rho}[x \mapsto \widehat{v}_1](m \uplus m_1), e_2, \widehat{\sigma}_1(m \uplus m_1))(s_2) \neq \bot$. Then, by definition of run, we have that

$$\mathsf{run}(\widehat{\rho}[x \mapsto \widehat{v}_1](m \uplus m_1), e_2, \widehat{\sigma}_1(m \uplus m_1))(s_2) = \langle \mathsf{dom}(\sigma_2) \setminus \mathsf{dom}(\widehat{\sigma}_1(m \uplus m_1)) \rangle (v_2, \sigma_2)$$

for some $(v_2, \sigma_2) \in \mathsf{Value} \times \mathsf{Store}$. Because of our assumption that

$$\mathsf{run}(\widehat{\rho}(m), e_1, \widehat{\sigma}(m))(s_1) = \langle \mathsf{dom}(\widehat{\sigma}_1(m \uplus m_1)) \setminus \mathsf{dom}(\widehat{\sigma}(m)) \rangle (\widehat{v}_1(m \uplus m_1), \widehat{\sigma}_1(m \uplus m_1))$$

there must exist a derivation in the concrete big-step operational semantics whose conclusion is

$$\widehat{\rho}(m), s_1 \vdash (e_1, \widehat{\sigma}(m)) \Downarrow (\widehat{v}_1(m \uplus m_1), \widehat{\sigma}_1(m \uplus m_1))$$

by the definition of run. Likewise, there must exist a derivation in the concrete big-step operational semantics whose conclusion is

$$\widehat{\rho}(m)[x \mapsto \widehat{v}_1(m \uplus m_1)], s_2 \vdash (e_2, \widehat{\sigma}_1(m \uplus m_1)) \Downarrow (v_2, \sigma_2)$$

using the fact that $\widehat{\rho}[x \mapsto \widehat{v}_1](m \uplus m_1) = \widehat{\rho}(m)[x \mapsto \widehat{v}_1(m \uplus m_1)]$. Hence, we can prove that

$$\widehat{\rho}(m), s \vdash (\mathtt{let}\ x = e_1\ \mathtt{in}\ e_2, \widehat{\sigma}(m)) \Downarrow (v_2, \sigma_2)$$

By the definition of run, we get

$$\mathsf{run}(\widehat{\rho}(m), \mathtt{let}\ x = e_1\ \mathtt{in}\ e_2, \widehat{\sigma}(m))(s) = \langle \mathsf{dom}(\sigma_2) \setminus \mathsf{dom}(\widehat{\sigma}(m)) \rangle (v_2, \sigma_2)$$

We must now show that

$$\langle \mathsf{dom}(\sigma_2) \setminus \mathsf{dom}(\widehat{\sigma}(m)) \rangle(v_2, \sigma_2) \in \mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), [\psi_1 : \widehat{v}_2]|_m, [\psi_1 : \widehat{\sigma}_2]|_m))$$

As in the previous case, we know that

$$\mathsf{img}(\mathsf{run}(\widehat{\rho}[x \mapsto \widehat{v}](m \uplus m_1), e_2, \widehat{\sigma}_1(m \uplus m_1))) = \mathsf{img}(\mathsf{result}_{V_2}(\widehat{\sigma}(m \uplus m_1), \widehat{v}_2|_{m \uplus m_1}, \widehat{\sigma}_2|_{m \uplus m_1}))$$

because $(\widehat{\rho}[x \mapsto \widehat{v}_1](m \uplus m_1), e_2, \widehat{\sigma}_1(m \uplus m_1)) \in \mathsf{Config}$. Thus, we get

$$\mathsf{result}_{V_2}(\widehat{\sigma}(m \uplus m_1), \widehat{v}_2|_{m \uplus m_1}, \widehat{\sigma}_2|_{m \uplus m_1})(m_2) = \langle \mathsf{dom}(\sigma_2) \setminus \mathsf{dom}(\widehat{\sigma}_1(m \uplus m_1)) \rangle(v_2, \sigma_2)$$

for some $m_2 \in \mathsf{Model}_{V_2}$ such that $v_2 = \widehat{v}_2(m \uplus m_1 \uplus m_2)$ and $\sigma_2 = \widehat{\sigma}(m \uplus m_1 \uplus m_2)$. Therefore, for some $m_2 \in \mathsf{Model}_{V_2}$, we get

$$[\psi_1 : \widehat{v}_2]|_m(m_1 \uplus m_2) = [\psi_1 : \widehat{v}_2](m \uplus m_1 \uplus m_2) = \widehat{v}_2(m \uplus m_1 \uplus m_2) = v_2$$
$$[\psi_1 : \widehat{\sigma}_2]|_m(m_1 \uplus m_2) = [\psi_1 : \widehat{\sigma}_2](m \uplus m_1 \uplus m_2) = \widehat{\sigma}_2(m \uplus m_1 \uplus m_2) = \sigma_2$$

meaning that

$$\mathsf{result}_V(\widehat{\sigma}(m), [\psi_1 : \widehat{v}_2]|_m, [\psi_1 : \widehat{\sigma}_2]|_m)(m_1 \uplus m_2) = \langle \mathsf{dom}(\sigma_2) \setminus \mathsf{dom}(\widehat{\sigma}(m)) \rangle(v_2, \sigma_2)$$

Therefore, we get that

$$\langle \mathsf{dom}(\sigma_2) \setminus \mathsf{dom}(\widehat{\sigma}(m)) \rangle(v_2, \sigma_2) \in \mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), [\psi_1 : \widehat{v}_2]|_m, [\psi_1 : \widehat{\sigma}_2]|_m))$$

The cases above collectively prove that

$$\mathsf{run}(\widehat{\rho}(m), \mathtt{let}\ x = e_1\ \mathtt{in}\ e_2, \widehat{\sigma}(m))(s) \in \mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), [\psi_1 : \widehat{v}_2]|_m, [\psi_1 : \widehat{\sigma}_2]|_m))$$

for all $s \in \mathbb{B}^{\mathbb{N}}$, meaning

$$\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), \mathtt{let}\ x = e_1\ \mathtt{in}\ e_2, \widehat{\sigma}(m))) \subseteq \mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), [\psi_1 : \widehat{v}_2]|_m, [\psi_1 : \widehat{\sigma}_2]|_m))$$

($\supseteq$) Take any $m' \in \mathsf{Model}_V$ and write $m' = m_1 \uplus m_2$ for $m_1 \in \mathsf{Model}_{V_1}$ and $m_2 \in \mathsf{Model}_{V_2}$. We will consider several cases for the evaluation of $\mathsf{result}_V(\widehat{\sigma}(m), [\psi_1 : \widehat{v}_2]|_m, [\psi_1 : \widehat{\sigma}_2]|_m)(m')$ and show that, in each case,

$$\mathsf{result}_V(\widehat{\sigma}(m), [\psi_1 : \widehat{v}_2]|_m, [\psi_1 : \widehat{\sigma}_2]|_m)(m') \in \mathsf{img}(\mathsf{run}(\widehat{\rho}(m), \mathtt{let}\ x = e_1\ \mathtt{in}\ e_2, \widehat{\sigma}(m)))$$

(a) Suppose that $\psi_1(m \uplus m_1) = \mathsf{F}$. We immediately get that

$$[\psi_1 : \widehat{v}_2]|_m(m') = [\psi_1 : \widehat{v}_2](m \uplus m_1 \uplus m_2) = \bot$$
$$[\psi_1 : \widehat{\sigma}_2]|_m(m') = [\psi_1 : \widehat{\sigma}_2](m \uplus m_1 \uplus m_2) = \bot$$

meaning that

$$\mathsf{result}_V(\widehat{\sigma}(m), [\psi_1 : \widehat{v}_2]|_m, [\psi_1 : \widehat{\sigma}_2]|_m)(m') = \bot$$

by definition.

We must now show that $\bot \in \mathsf{img}(\mathsf{run}(\widehat{\rho}(m), \mathtt{let}\ x = e_1\ \mathtt{in}\ e_2, \widehat{\sigma}(m)))$. Because we have that $(\widehat{\rho}(m), e_1, \widehat{\sigma}(m)) \in \mathsf{Config}$, we get that

$$\psi_1(m \uplus m_1) = \mathsf{T} \iff \widehat{v}_1(m \uplus m_1) \neq \bot \text{ and } \widehat{\sigma}_1(m \uplus m_1) \neq \bot$$

using the inductive hypothesis. The assumption that $\psi_1(m \uplus m_1) = \mathsf{F}$ implies that at least one of $\widehat{v}_1(m \uplus m_1) \neq \bot$ or $\widehat{\sigma}_1(m \uplus m_1) \neq \bot$. Then, by definition of result_{V_1}, we get that

$$\mathsf{result}_{V_1}(\widehat{\rho}(m), \widehat{v}_1|_m, \widehat{\sigma}_1|_m)(m_1) = \bot$$

Applying the inductive hypothesis again, we have that

$$\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), e_1, \widehat{\sigma}(m))) = \mathsf{img}(\mathsf{result}_{V_1}(\widehat{\sigma}(m), \widehat{v}_1|_m, \widehat{\sigma}_1|_m))$$

Hence, because $\mathsf{result}_{V_1}(\widehat{\rho}(m), \widehat{v}_1|_m, \widehat{\sigma}_1|_m)(m_1) = \bot$, we get $\mathsf{run}(\widehat{\rho}(m), e_1, \widehat{\sigma}(m))(s_1) = \bot$ for some $s_1 \in \mathbb{B}^{\mathbb{N}}$. Now, construct $s \in \mathbb{B}^{\mathbb{N}}$ with $s \rightsquigarrow s_1, s_2$ for some other $s_2 \in \mathbb{B}^{\mathbb{N}}$. It follows that

$\mathsf{run}(\widehat{\rho}(m), \mathtt{let}\ x = e_1\ \mathtt{in}\ e_2)(s) = \bot$ (see case (a) of the ($\subseteq$) argument for a thorough proof by contradiction). We conclude that

$$\bot \in \mathsf{img}(\mathsf{run}(\widehat{\rho}(m), \mathtt{let}\ x = e_1\ \mathtt{in}\ e_2, \widehat{\sigma}(m)))$$

(b) Suppose that $\psi_1(m \uplus m_1) = \mathsf{T}$. Recall, by the inductive hypothesis, we have that

$$\psi_1(m \uplus m_1) = \mathsf{T} \iff \widehat{v}_1(m \uplus m_1) \neq \bot \text{ and } \widehat{\sigma}_1(m \uplus m_1) \neq \bot$$

Hence, $\psi_1(m \uplus m_1) = \mathsf{T}$ implies that

$$\mathsf{result}_{V_1}(\widehat{\sigma}(m), \widehat{v}_1|_m, \widehat{\sigma}_1|_m)(m_1) = \langle \mathsf{dom}(\widehat{\sigma}_1(m \uplus m_1)) \setminus \mathsf{dom}(\widehat{\sigma}(m)) \rangle (\widehat{v}_1(m \uplus m_1), \widehat{\sigma}_1(m \uplus m_1))$$

Again by the inductive hypothesis, we have that

$$\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), e_1, \widehat{\sigma}(m))) = \mathsf{img}(\mathsf{result}_{V_1}(\widehat{\sigma}(m), \widehat{v}_1|_m, \widehat{\sigma}_1|_m))$$

Therefore, there exists some $s_1 \in \mathbb{B}^{\mathbb{N}}$ such that

$$\mathsf{run}(\widehat{\rho}(m), e_1, \widehat{\sigma}(m))(s_1) = \langle \mathsf{dom}(\widehat{\sigma}_1(m \uplus m_1)) \setminus \mathsf{dom}(\widehat{\sigma}(m)) \rangle (\widehat{v}_1(m \uplus m_1), \widehat{\sigma}_1(m \uplus m_1))$$

which, by definition, implies that there exists a derivation in the concrete big-step operational semantics whose conclusion is

$$\widehat{\rho}(m), s_1 \vdash (e_1, \widehat{\sigma}(m)) \Downarrow (\widehat{v}_1(m \uplus m_1), \widehat{\sigma}_1(m \uplus m_1))$$

As in the ($\subseteq$) proof, we can show

$$(\widehat{\rho}(m)[x \mapsto \widehat{v}_1(m \uplus m_1)], e_1, \widehat{\sigma}_1(m \uplus m_1)) = (\widehat{\rho}[x \mapsto \widehat{v}_1](m \uplus m_1), e_1, \widehat{\sigma}_1(m \uplus m_1)) \in \mathsf{Config}$$

as a direct corollary of Theorem B.14. We must now case on the evaluation of $\widehat{v}_2(m \uplus m')$ and $\widehat{\sigma}_2(m \uplus m')$:

(i) Suppose at least one of $\widehat{v}_2(m \uplus m') = \bot$ or $\widehat{\sigma}_2(m \uplus m') = \bot$. If $\widehat{v}_2(m \uplus m') = \bot$, then

$$[\psi_1 : \widehat{v}_2]_m(m') = [\psi_1 : \widehat{v}_2](m \uplus m') = \widehat{v}_2(m \uplus m') = \bot$$

meaning that

$$\mathsf{result}_V(\widehat{\sigma}(m), [\psi_1 : \widehat{v}_2]|_m, [\psi_1 : \widehat{\sigma}_2]|_m)(m') = \bot$$

by definition. Likewise, if $\widehat{\sigma}_2(m \uplus m') = \bot$, then

$$[\psi_1 : \widehat{\sigma}_2]_m(m') = [\psi_1 : \widehat{\sigma}_2](m \uplus m') = \widehat{\sigma}_2(m \uplus m') = \bot$$

meaning that

$$\mathsf{result}_V(\widehat{\sigma}(m), [\psi_1 : \widehat{v}_2]|_m, [\psi_1 : \widehat{\sigma}_2]|_m)(m') = \bot$$

by definition. In either case, $\mathsf{result}_V(\widehat{\sigma}(m), [\psi_1 : \widehat{v}_2]|_m, [\psi_1 : \widehat{\sigma}_2]|_m)(m') = \bot$, so it suffices to show that $\bot \in \mathsf{img}(\mathsf{run}(\widehat{\rho}(m), \mathtt{let}\ x = e_1\ \mathtt{in}\ e_2, \widehat{\sigma}(m)))$.
Because $(\widehat{\rho}[x \mapsto \widehat{v}_1](m \uplus m_1), e_1, \widehat{\sigma}_1(m \uplus m_1)) \in \mathsf{Config}$, we have that

$$\mathsf{img}(\mathsf{run}(\widehat{\rho}[x \mapsto \widehat{v}](m \uplus m_1), e_2, \widehat{\sigma}_1(m \uplus m_1))) = \mathsf{img}(\mathsf{result}_{V_2}(\widehat{\sigma}(m \uplus m_1), \widehat{v}_2|_{m \uplus m_1}, \widehat{\sigma}_2|_{m \uplus m_1}))$$

using the inductive hypothesis. Therefore, there exists $s_2 \in \mathbb{B}^{\mathbb{N}}$, such that

$$\mathsf{run}(\widehat{\rho}[x \mapsto \widehat{v}](m \uplus m_1), e_2, \widehat{\sigma}_1(m \uplus m_1))(s_2) = \bot$$

Now, construct $s \in \mathbb{B}^{\mathbb{N}}$ such that $s \rightsquigarrow s_1, s_2$. Because we have shown

$$\mathsf{run}(\widehat{\rho}(m), e_1, \widehat{\sigma}(m))(s_1) = \langle \mathsf{dom}(\widehat{\sigma}_1(m \uplus m_1)) \setminus \mathsf{dom}(\widehat{\sigma}(m)) \rangle (\widehat{v}_1(m \uplus m_1), \widehat{\sigma}_1(m \uplus m_1)) \neq \bot$$
$$\mathsf{run}(\widehat{\rho}[x \mapsto \widehat{v}](m \uplus m_1), e_2, \widehat{\sigma}_1(m \uplus m_1))(s_2) = \bot$$

it follows that $\mathsf{run}(\widehat{\rho}(m), \texttt{let } x = e_1 \texttt{ in } e_2, \widehat{\sigma}(m))(s) = \bot$ (see case (b) of the ($\subseteq$) argument for detailed proof). Therefore,

$$\bot \in \mathsf{img}(\mathsf{run}(\widehat{\rho}(m), \texttt{let } x = e_1 \texttt{ in } e_2, \widehat{\sigma}(m)))$$

(ii) Suppose $\widehat{v}_2(m \uplus m') \neq \bot$ and $\widehat{\sigma}_2(m \uplus m') \neq \bot$. Then we have

$$[\psi_1 : \widehat{v}_2]|_m(m') = [\psi_1 : \widehat{v}_2](m \uplus m') = \widehat{v}_2(m \uplus m') \neq \bot$$
$$[\psi_1 : \widehat{\sigma}_2]|_m(m') = [\psi_1 : \widehat{\sigma}_2](m \uplus m') = \widehat{\sigma}_2(m \uplus m') \neq \bot$$

meaning

$$\mathsf{result}_V(\widehat{\sigma}(m), [\psi_1 : \widehat{v}_2]|_m, [\psi_1 : \widehat{\sigma}_2]|_m)(m') = \langle L \rangle(\widehat{v}_2(m \uplus m'), \widehat{\sigma}_2(m \uplus m'))$$

where $L = \mathsf{dom}(\widehat{\sigma}_2(m \uplus m')) \setminus \mathsf{dom}(\widehat{\sigma}(m))$ by definition of result_V. It then suffices to show that $\langle L \rangle(\widehat{v}_2(m \uplus m'), \widehat{\sigma}_2(m \uplus m')) \in \mathsf{img}(\mathsf{run}(\widehat{\rho}(m), \texttt{let } x = e_1 \texttt{ in } e_2, \widehat{\sigma}(m)))$.
As previously stated, we know that there exists some $s_1 \in \mathbb{B}^{\mathbb{N}}$ such that

$$\mathsf{run}(\widehat{\rho}(m), e_1, \widehat{\sigma}(m))(s_1) = \langle \mathsf{dom}(\widehat{\sigma}_1(m \uplus m_1)) \setminus \mathsf{dom}(\widehat{\sigma}(m)) \rangle(\widehat{v}_1(m \uplus m_1), \widehat{\sigma}_1(m \uplus m_1))$$

and, subsequently, that

$$\widehat{\rho}(m), s_1 \vdash (e_1, \widehat{\sigma}(m)) \Downarrow (\widehat{v}_1(m \uplus m_1), \widehat{\sigma}_1(m \uplus m_1))$$

By the inductive hypothesis, we have that

$$\mathsf{img}(\mathsf{run}(\widehat{\rho}[x \mapsto \widehat{v}](m \uplus m_1), e_2, \widehat{\sigma}_1(m \uplus m_1))) = \mathsf{img}(\mathsf{result}_{V_2}(\widehat{\sigma}(m \uplus m_1), \widehat{v}_2|_{m \uplus m_1}, \widehat{\sigma}_2|_{m \uplus m_1}))$$

This implies that there exists $s_2 \in \mathbb{B}^{\mathbb{N}}$ such that

$$\mathsf{run}(\widehat{\rho}[x \mapsto \widehat{v}](m \uplus m_1), e_2, \widehat{\sigma}_1(m \uplus m_1))(s_2) = \langle L \rangle(\widehat{v}_2(m \uplus m'), \widehat{\sigma}_2(m \uplus m'))$$

meaning that there exists a derivation in the concrete big-step operational semantics whose conclusion is

$$\widehat{\rho}(m)[x \mapsto \widehat{v}(m \uplus m_1)], s_2 \vdash (e_2, \widehat{\sigma}_1(m \uplus m_1)) \Downarrow (\widehat{v}_2(m \uplus m'), \widehat{\sigma}_2(m \uplus m'))$$

using the fact that $\widehat{\rho}[x \mapsto \widehat{v}](m \uplus m_1) = \widehat{\rho}(m)[x \mapsto \widehat{v}(m \uplus m_1)]$. Now, construct $s \in \mathbb{B}^{\mathbb{N}}$ with $s \rightsquigarrow s_1, s_2$. We can prove that

$$\widehat{\rho}(m), s \vdash (\texttt{let } x = e_1 \texttt{ in } e_2, \widehat{\sigma}(m)) \Downarrow (\widehat{v}_2(m \uplus m'), \widehat{\sigma}_2(m \uplus m'))$$

using the concrete big-step judgement Let. Hence, by definition of run, we get that

$$\mathsf{run}(\widehat{\rho}(m), \texttt{let } x = e_1 \texttt{ in } e_2, \widehat{\sigma}(m))(s) = \langle L \rangle(\widehat{v}_2(m \uplus m'), \widehat{\sigma}_2(m \uplus m'))$$

meaning $\langle L \rangle(\widehat{v}_2(m \uplus m'), \widehat{\sigma}_2(m \uplus m')) \in \mathsf{img}(\mathsf{run}(\widehat{\rho}(m), \texttt{let } x = e_1 \texttt{ in } e_2, \widehat{\sigma}(m)))$.

The cases above collectively imply that

$$\mathsf{result}_V(\widehat{\sigma}(m), [\psi_1 : \widehat{v}_2]|_m, [\psi_1 : \widehat{\sigma}_2]|_m)(m') \in \mathsf{img}(\mathsf{run}(\widehat{\rho}(m), \texttt{let } x = e_1 \texttt{ in } e_2, \widehat{\sigma}(m)))$$

for all $m' \in \mathsf{Model}_V$, meaning

$$\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), \texttt{let } x = e_1 \texttt{ in } e_2, \widehat{\sigma}(m))) \supseteq \mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), [\psi_1 : \widehat{v}_2]|_m, [\psi_1 : \widehat{\sigma}_2]|_m))$$

We then conclude that

$$\mathsf{img}(\mathsf{run}(\widehat{\rho}(m), \texttt{let } x = e_1 \texttt{ in } e_2, \widehat{\sigma}(m))) = \mathsf{img}(\mathsf{result}_V(\widehat{\sigma}(m), [\psi_1 : \widehat{v}_2]|_m, [\psi_1 : \widehat{\sigma}_2]|_m))$$

completing our proof of (1).

Now, we turn our attention to (2). It suffices to show that

$$(\psi_1 \wedge \psi_2)(m \uplus m') = \mathsf{T} \iff [\psi_1 : \widehat{v}_2](m \uplus m') \neq \bot \text{ and } [\psi_1 : \widehat{\sigma}_2](m \uplus m') \neq \bot$$

for all $m' \in \mathsf{Model}_V$. Write $m' = m_1 \uplus m_2$ for $m_1 \in \mathsf{Model}_{V_1}$ and $m_2 \in \mathsf{Model}_{V_2}$.

($\Longrightarrow$) Suppose that $(\psi_1 \wedge \psi_2)(m \uplus m') = \mathsf{T}$. This means that $\psi_1(m \uplus m_1) = \mathsf{T}$ and $\psi_2(m \uplus m') = \mathsf{T}$. By the inductive hypothesis, we have

$$\psi_2(m \uplus m_1 \uplus m_2) = \mathsf{T} \Longleftrightarrow \widehat{v}_2(m \uplus m_1 \uplus m_2) \neq \bot \text{ and } \widehat{\sigma}_2(m \uplus m_1 \uplus m_2) \neq \bot$$

as $(m \uplus m_1) \in \mathsf{Model}_{\mathsf{symvars}(\widehat{\rho}[x \mapsto \widehat{v}_1], \widehat{\sigma}_1)}$ and $m_2 \in \mathsf{Model}_{V_2}$ by construction. Hence, we must have that $\widehat{v}_2(m \uplus m') \neq \bot$ and $\widehat{\sigma}_2(m \uplus m') \neq \bot$. Therefore,

$$[\psi_1 : \widehat{v}_2](m \uplus m') = \widehat{v}_2(m \uplus m') \neq \bot$$
$$[\psi_1 : \widehat{\sigma}_2](m \uplus m') = \widehat{\sigma}_2(m \uplus m') \neq \bot$$

as desired.

($\Longleftarrow$) Suppose $[\psi_1 : \widehat{v}_2](m \uplus m') \neq \bot$ and $[\psi_1 : \widehat{\sigma}_2](m \uplus m') \neq \bot$. Repeat the proof of ($\Longrightarrow$) in the opposite direction.

This proves (2).

This completes our proof by induction. □

Definition B.22. The set of *answers*, denoted Ans, are defined to be

$$\mathsf{Ans} ::= () \mid b \mid r \mid (a, a) \mid \mathsf{opaque}$$

Definition B.23. Let $\lfloor \cdot \rfloor$ be the function

$$\lfloor \cdot \rfloor : \mathsf{Value} \to \mathsf{Ans}$$
$$\lfloor v \rfloor = \begin{cases} v & \text{if } v \in \mathsf{Bool} \cup \mathsf{Num} \cup \{()\} \\ (\lfloor v_1 \rfloor, \lfloor v_2 \rfloor) & \text{if } v = (v_1, v_2) \\ \mathsf{opaque} & \text{otherwise} \end{cases}$$

Definition B.24. Let eval be the function

$$\mathsf{eval} : \mathsf{Expr} \to (\mathsf{Stream} \to \mathsf{Ans}_\bot)$$
$$\mathsf{eval}(e)(s) = \begin{cases} \lfloor v \rfloor & \text{if } \varnothing, s \vdash (e, \varnothing) \Downarrow (v, \sigma) \\ \bot & \text{otherwise} \end{cases}$$

and ans be the dependently-typed function

$$\mathsf{ans} : (\widehat{\mathsf{Value}} \times \widehat{\mathsf{Store}}) \to (\mathsf{Model}_V \to \mathsf{Ans}_\bot)$$
$$\mathsf{ans}(\widehat{v}, \widehat{\sigma})(m) = \begin{cases} \lfloor \widehat{v}(m) \rfloor & \text{if } \widehat{v}(m) \neq \bot, \widehat{\sigma}(m) \neq \bot \\ \bot & \text{otherwise} \end{cases} \text{ where } V = \mathsf{symvars}(\widehat{v}, \widehat{\sigma})$$

Theorem B.25 (Correctness of Closed Expressions in Idealized Rosette)**.** Let $e \in \mathsf{Expr}$ such that $\mathsf{FV}(e) = \varnothing$ and $\varnothing \vdash (e, \varnothing) \Downarrow\!\Downarrow (\widehat{v}, \widehat{\sigma}, \psi)$. Then

(1) $\mathsf{img}(\mathsf{eval}(e)) = \mathsf{img}(\mathsf{ans}(\widehat{v}, \widehat{\sigma}))$;
(2) $\psi(m) = \mathsf{T} \Longleftrightarrow \widehat{v}(m) \neq \bot$ and $\widehat{\sigma}(m) \neq \bot$, for all $m \in \mathsf{Model}_{\mathsf{symvars}(\widehat{v}, \widehat{\sigma})}$.

Proof. Observe that $(\varnothing, e, \varnothing) \in \mathsf{Config}$ due to the fact that $\mathsf{FV}(e) = \varnothing$. Moreover, we have that $\mathsf{Model}_{(\varnothing, \varnothing)} = \{\varnothing\}$. Then, by Theorem B.21, we get

$$\mathsf{img}(\mathsf{run}(\varnothing, e, \varnothing)) = \mathsf{img}(\mathsf{result}_V(\varnothing, \widehat{v}|_\varnothing, \widehat{\sigma}|_\varnothing))$$

and

$$\psi(\varnothing \uplus m) = \mathsf{T} \Longleftrightarrow \widehat{v}(\varnothing \uplus m) \neq \bot \text{ and } \widehat{\sigma}(\varnothing \uplus m), \text{ for all } m \in \mathsf{Model}_V$$

where $V = \mathsf{symvars}(\widehat{v}, \widehat{\sigma})$. The second point immediately implies (2). We break the proof of (1) into two steps:

($\subseteq$) Consider $s \in \mathbb{B}^{\mathbb{N}}$. We must show that $\mathsf{eval}(e)(s) \in \mathsf{img}(\mathsf{ans}(\widehat{v}, \widehat{\sigma}))$. There are two cases.
If $\varnothing, s, \vdash (e, \varnothing) \Downarrow (v, \sigma)$, then $\mathsf{eval}(e)(s) = \lfloor v \rfloor$. Likewise, $\mathsf{run}(\varnothing, e, \varnothing)(s) = \langle \mathsf{dom}(\sigma) \rangle (v, \sigma)$. Because $\mathsf{img}(\mathsf{run}(\varnothing, e, \varnothing)) = \mathsf{img}(\mathsf{result}_V(\varnothing, \widehat{v}|_\varnothing, \widehat{\sigma}|_\varnothing))$, there must then exist some $m \in \mathsf{Model}_{\mathsf{symvars}(\widehat{v}, \widehat{\sigma})}$ such that $\mathsf{result}_V(\varnothing, \widehat{v}|_\varnothing, \widehat{\sigma}|_\varnothing)(m) = \langle \mathsf{dom}(\sigma) \rangle (v, \sigma)$ where $v = \widehat{v}(m)$ and $\sigma = \widehat{\sigma}(m)$. Hence, $\mathsf{ans}(\widehat{v}, \widehat{\sigma})(m) = \lfloor \widehat{v}(m) \rfloor = \lfloor v \rfloor$. Therefore, $\mathsf{eval}(e)(s) \in \mathsf{img}(\mathsf{ans}(\widehat{v}, \widehat{\sigma}))$.

On the other hand, if there is not derivation in the concrete big-step operational semantics whose conclusion is $\varnothing, s, \vdash (e, \varnothing) \Downarrow (v, \sigma)$, then $\mathsf{eval}(e)(s) = \bot$. Thus, $\mathsf{run}(\varnothing, e, \varnothing)(s) = \bot$, and we conclude that $\mathsf{result}_V(\varnothing, \widehat{v}|_\varnothing, \widehat{\sigma}|_\varnothing)(m) = \bot$ for some $m \in \mathsf{Model}_{\mathsf{symvars}(\widehat{v}, \widehat{\sigma})}$. By definition of result_V, this can only occur if $\widehat{v}(m) = \bot$ or $\widehat{\sigma}(m) = \bot$. In this case, we also have that $\mathsf{ans}(\widehat{v}, \widehat{\sigma})(m) = \bot$. Hence, $\mathsf{eval}(e)(s) \in \mathsf{img}(\mathsf{ans}(\widehat{v}, \widehat{\sigma}))$, implying that $\mathsf{img}(\mathsf{eval}(e)) \subseteq \mathsf{img}(\mathsf{ans}(\widehat{v}, \widehat{\sigma}))$.

($\supseteq$) Consider $m \in \mathsf{Model}_{\mathsf{symvars}(\widehat{v}, \widehat{\sigma})}$. We must show that $\mathsf{ans}(\widehat{v}, \widehat{\sigma})(m) \in \mathsf{img}(\mathsf{eval}(e))$. There are two cases.
If $\widehat{v}(m) \neq \bot$ or $\widehat{\sigma}(m) \neq \bot$, then $\mathsf{ans}(\widehat{v}, \widehat{\sigma})(m) = \lfloor \widehat{v}(m) \rfloor$. Additionally, we know that $\mathsf{result}_V(\varnothing, \widehat{v}|_\varnothing, \widehat{\sigma}|_\varnothing)(m) = \langle \mathsf{dom}(\widehat{\sigma}(m)) \rangle (\widehat{v}(m), \widehat{\sigma}(m))$. Because we have $\mathsf{img}(\mathsf{run}(\varnothing, e, \varnothing)) = \mathsf{img}(\mathsf{result}_V(\varnothing, \widehat{v}|_\varnothing, \widehat{\sigma}|_\varnothing))$, there must then exist some $s \in \mathbb{B}^{\mathbb{N}}$ such that $\mathsf{run}(\varnothing, e, \varnothing)(s) = \langle \mathsf{dom}(\widehat{\sigma}(m)) \rangle (\widehat{v}(m), \widehat{\sigma}(m))$. This is only the case when $\varnothing, s \vdash (e, \varnothing) \Downarrow (\widehat{v}(m), \widehat{\sigma}(m))$. Hence, we get $\mathsf{eval}(e)(s) = \lfloor \widehat{v}(m) \rfloor$, meaning $\mathsf{ans}(\widehat{v}, \widehat{\sigma})(m) \in \mathsf{img}(\mathsf{eval}(e))$.

Alternatively, suppose that $\widehat{v}(m) = \bot$ or $\widehat{\sigma}(m) = \bot$. Then $\mathsf{ans}(\widehat{v}, \widehat{\sigma})(m) = \bot$. Further, $\mathsf{result}_V(\varnothing, \widehat{v}|_\varnothing, \widehat{\sigma}|_\varnothing)(m) = \bot$. We conclude that $\mathsf{run}(\varnothing, e, \varnothing)(s) = \bot$ for some $s \in \mathbb{B}^{\mathbb{N}}$, meaning that there does not exist a derivation in the concrete big-step operational semantics whose conclusion is $\varnothing, s \vdash (e, \varnothing) \Downarrow (v, \sigma)$ for some $(v, \sigma) \in \mathsf{Value} \times \mathsf{Store}$. Hence, we also have $\mathsf{eval}(e)(s) = \bot$. This implies $\mathsf{ans}(\widehat{v}, \widehat{\sigma})(m) \in \mathsf{img}(\mathsf{eval}(e))$, and we have $\mathsf{img}(\mathsf{eval}(e)) \supseteq \mathsf{img}(\mathsf{ans}(\widehat{v}, \widehat{\sigma}))$.

This completes the proof. □

C Idealized Roulette

C.1 Syntax

$$
\begin{aligned}
e \in \mathsf{Expr} ::= {} & x \mid \mathtt{let}\ x = e\ \mathtt{in}\ e \mid \lambda x.e \mid x\ y \\
& \mid \mathtt{true} \mid \mathtt{false} \mid \mathtt{if}\ x\ e\ e \\
& \mid r \mid x + y \mid x - y \mid x \times y \\
& \mid () \mid (x, y) \mid \mathtt{fst}\, x \mid \mathtt{snd}\, x \\
& \mid \mathtt{ref}\, x \mid\ !y \mid x := y \\
& \mid \mathtt{flip}\, x \mid \mathtt{fail} \\
\rho \in \mathsf{Env} := {} & \mathsf{Var} \to_{\mathsf{fin}} \mathsf{Val} \\
v \in \mathsf{Val} ::= {} & \mathtt{true} \mid \mathtt{false} \mid \mathtt{clo}(\lambda x.e, \rho) \mid r \mid () \mid (v, v) \mid \ell \in \mathsf{Loc} \\
\sigma \in \mathsf{Store} := {} & \mathsf{Loc} \to_{\mathsf{fin}} \mathsf{Val} \\
r \in {} & \mathbb{Q} \\
s \in \mathsf{EntropySrc} := {} & \mathsf{Stream}[0,1] \cong [0,1]^{\mathbb{N}}
\end{aligned}
$$

$$\boxed{\rho, s \vdash (e, \sigma) \Downarrow (v, \sigma')}$$

$$\textsc{Var}\ \frac{x \in \mathrm{dom}(\rho)}{\rho, s \vdash (x, \sigma) \Downarrow (\rho(x), \sigma)} \qquad \textsc{Let}\ \frac{s \rightsquigarrow s_1, s_2 \quad \rho, s_1 \vdash (e_1, \sigma) \Downarrow (v_1, \sigma_1) \quad \rho[x \mapsto v_1], s_2 \vdash (e_2, \sigma_1) \Downarrow (v, \sigma')}{\rho, s \vdash (\mathtt{let}\ x = e_1\ \mathtt{in}\ e_2, \sigma) \Downarrow (v, \sigma')} \qquad \textsc{Lam}\ \frac{}{\rho, s \vdash (\lambda x.e, \sigma) \Downarrow (\mathtt{clo}(\lambda x.e, \rho), \sigma)}$$

$$\textsc{App}\ \frac{\rho(x_1) = \mathtt{clo}(\lambda x'.e', \rho') \quad \rho'[x' \mapsto \rho(x_2)], s \vdash (e', \sigma) \Downarrow (v, \sigma')}{\rho, s \vdash (x_1 x_2, \sigma) \Downarrow (v, \sigma')} \qquad \textsc{True}\ \frac{}{\rho, s \vdash (\mathtt{true}, \sigma) \Downarrow (\mathtt{true}, \sigma)} \qquad \textsc{False}\ \frac{}{\rho, s \vdash (\mathtt{false}, \sigma) \Downarrow (\mathtt{false}, \sigma)}$$

$$\textsc{IfTrue}\ \frac{x \in \mathrm{dom}(\rho) \quad \rho(x) = \mathtt{true} \quad \rho, s \vdash (e_1, \sigma) \Downarrow (v, \sigma')}{\rho, s \vdash (\mathtt{if}\ x\ e_1\ e_2, \sigma) \Downarrow (v, \sigma')} \qquad \textsc{IfFalse}\ \frac{x \in \mathrm{dom}(\rho) \quad \rho(x) = \mathtt{false} \quad \rho, s \vdash (e_2, \sigma) \Downarrow (v, \sigma')}{\rho, s \vdash (\mathtt{if}\ x\ e_1\ e_2, \sigma) \Downarrow (v, \sigma')} \qquad \textsc{Num}\ \frac{r \in \mathbb{Q}}{\rho, s \vdash (r, \sigma) \Downarrow (r, \sigma)}$$

$$\textsc{Arith}\ \frac{x_1 \in \mathrm{dom}(\rho) \quad x_2 \in \mathrm{dom}(\rho) \quad \rho(x_1), \rho(x_2) \in \mathbb{Q} \quad \oplus \in \{+, -, \times\}}{\rho, s \vdash (x_1 \oplus x_2, \sigma) \Downarrow (\rho(x_1)\ [\![\oplus]\!]\ \rho(x_2), \sigma)} \qquad \textsc{Pair}\ \frac{x, y \in \mathrm{dom}(\rho)}{\rho, s \vdash ((x, y), \sigma) \Downarrow ((\rho(x), \rho(y)), \sigma)}$$

$$\textsc{Fst}\ \frac{\rho(x) = (v, w)}{\rho, s \vdash (\mathtt{fst}\, x, \sigma) \Downarrow (v, \sigma)} \qquad \textsc{Snd}\ \frac{\rho(x) = (v, w)}{\rho, s \vdash (\mathtt{snd}\, x, \sigma) \Downarrow (w, \sigma)} \qquad \textsc{Ref}\ \frac{x \in \mathrm{dom}(\rho) \quad \ell \notin \mathrm{locs}(\rho, \sigma)}{\rho, s \vdash (\mathtt{ref}\, x, \sigma) \Downarrow (\ell, \sigma \uplus \{\ell \mapsto \rho(x)\})}$$

$$\textsc{Get}\ \frac{x \in \mathrm{dom}(\rho) \quad \rho(x) \in \mathrm{dom}(\sigma)}{\rho, s \vdash (!x, \sigma) \Downarrow (\sigma(\rho(x)), \sigma)} \qquad \textsc{Set}\ \frac{x \in \mathrm{dom}(\rho) \quad \rho(x) \in \mathrm{dom}(\sigma)}{\rho, s \vdash (x := y, \sigma) \Downarrow ((), \sigma[\rho(x) \mapsto \rho(y)])}$$

$$\textsc{FlipTrue}\ \frac{x \in \mathrm{dom}(\rho) \quad r < \rho(x)}{\rho, r :: s \vdash (\mathtt{flip}\, x, \sigma) \Downarrow (\mathtt{true}, \sigma)} \qquad \textsc{FlipFalse}\ \frac{x \in \mathrm{dom}(\rho) \quad r \geq \rho(x)}{\rho, r :: s \vdash (\mathtt{flip}\, x, \sigma) \Downarrow (\mathtt{false}, \sigma)}$$

Fig. 17. Concrete semantics.

C.2 Concrete semantics

Lemma C.1 (Invariant of the concrete semantics)**.** If $(\rho, e, \sigma) \in \mathrm{Config}$ and $\rho, s \vdash (e, \sigma) \Downarrow (v, \sigma')$, then $\sigma' = \sigma_{\mathrm{old}} \uplus \sigma_{\mathrm{new}}$ for some $\sigma_{\mathrm{old}}, \sigma_{\mathrm{new}}$ with $\mathrm{dom}(\sigma') = \mathrm{dom}(\sigma_{\mathrm{old}})$.

Proof. Analogous to Theorem B.14. □

Lemma C.2 (Nominal determinism)**.** If $(\rho, e, \sigma) \in \mathrm{Config}$ and $\rho, s \vdash (e, \sigma) \Downarrow (v_1, \sigma_1)$ and $\rho, s \vdash (e, \sigma) \Downarrow (v_2, \sigma_2)$ then $\langle \mathrm{dom}(\sigma_1) \setminus \mathrm{dom}(\sigma) \rangle (v_1, \sigma_1) = \langle \mathrm{dom}(\sigma_2) \setminus \mathrm{dom}(\sigma) \rangle (v_2, \sigma_2)$.

Proof. Analogous to Theorem B.17. □

Definition C.3. Let run be the function

$$\mathrm{run} : \mathrm{Config} \to \mathrm{EntropySrc} \to \mathrm{Result}_\perp$$

$$\mathrm{run}(\rho, e, \sigma)(s) = \begin{cases} \langle \mathrm{dom}(\sigma') \setminus \mathrm{dom}(\sigma) \rangle (v, \sigma') & \text{if } \rho, s \vdash (e, \sigma) \Downarrow (v, \sigma') \\ \perp, & \text{otherwise} \end{cases},$$

well-defined by Theorem C.2.

Lemma C.4. For all $(\rho, e, \sigma) \in \mathsf{Config}$ the partial application

$$\mathsf{run}(\rho, e, \sigma) : \mathsf{EntropySrc} \to \mathsf{Result}_\perp$$
$$\mathsf{run}(\rho, e, \sigma) = s \mapsto \mathsf{run}(\rho, s, e, \sigma)$$

is measurable as a function $[0,1]^{\mathbb{N}} \to \mathsf{Result}_\perp$, where $[0,1]^{\mathbb{N}}$ is given the σ-algebra induced by its usual Lebesgue measure as in Wand et al. [63], and $\mathsf{Result}_\perp$ is given the discrete σ-algebra.

Proof. Fix ρ, e, σ and a result r, with aim to show $\mathsf{run}(\rho, e, \sigma)^{-1}\{r\}$ is a measurable subset of $[0,1]^{\mathbb{N}}$. The relation $-,- \vdash (-,-) \Downarrow (-,-)$ is the least fixed point of a continuous functional F on $\wp(\mathsf{Env} \times \mathsf{EntropySpc} \times \mathsf{Exp} \times \mathsf{Store} \times \mathsf{Val} \times \mathsf{Store})$, so equal to the countable union of the sequence $(F^i(\varnothing))_{i \in \mathbb{N}}$, and one has $\rho, s \vdash (e, \sigma) \Downarrow (v, \sigma')$ iff $(\rho, s, e, \sigma, v, \sigma') \in F^n(\varnothing)$ for some n. It holds by induction on n that $\{s \mid (\rho, s, e, \sigma, r) \in F^n(\varnothing)\}$ is a measurable subset of $[0,1]^{\mathbb{N}}$ for all n, because the action of F can be written using countable unions, countable intersections, complements, and preimages of the measurable functions $\mathsf{split} : [0,1]^{\mathbb{N}} \to [0,1]^{\mathbb{N}} \times [0,1]^{\mathbb{N}}$ and $\mathsf{hd}(-) < a : [0,1]^{\mathbb{N}} \to \mathsf{Bool}$. From this it follows that $\{s \mid \rho, s \vdash (e, \sigma) \Downarrow (v, \sigma)\}$ is a countable union of measurable sets, hence measurable. Each $\mathsf{run}(\rho, e, \sigma)^{-1}\{r\}$ is then a union of sets of the form $\{s \mid \rho, s \vdash (e, \sigma) \Downarrow (v, \sigma)\}$, where ρ, v, σ range over the representatives of the equivalence class r; since there are only countably many representatives, this union is also countable, so $\mathsf{run}(\rho, e, \sigma)^{-1}\{r\}$ is measurable too. □

C.3 Abstract semantics

$$w \in \mathsf{WeightMap} := \mathsf{SymVar} \to_{\mathsf{fin}} [0,1]$$

Lemma C.5 (Determinism). If $\widehat{\rho} \vdash (e, \widehat{\sigma}, w) \Downarrow\!\Downarrow (\widehat{v}_1, \widehat{\sigma}_1, w_1, \psi_1)$ and $\widehat{\rho} \vdash (e, \widehat{\sigma}, w) \Downarrow\!\Downarrow (\widehat{v}_2, \widehat{\sigma}_2, w_2, \psi_2)$ then $(\widehat{v}_1, \widehat{\sigma}_1, w_1, \psi_1) = (\widehat{v}_2, \widehat{\sigma}_2, w_2, \psi_2)$.

Proof. By induction on $\widehat{\rho} \vdash (e, \widehat{\sigma}, w) \Downarrow\!\Downarrow (\widehat{v}_1, \widehat{\sigma}_1, w_1, \psi_1)$. □

Lemma C.6 (Invariant of the abstract semantics). If $\widehat{\rho} \vdash (e, \widehat{\sigma}, w) \Downarrow\!\Downarrow (\widehat{v}, \widehat{\sigma}', w', \psi)$, then:

(1) $w' = w \uplus w_{\mathsf{new}}$ for some w_{new};
(2) If $\mathsf{symvars}(\widehat{\rho}, \widehat{\sigma}) \subseteq A \subseteq \mathsf{dom}(w)$, then $\mathsf{symvars}(\widehat{v}, \widehat{\sigma}') \subseteq A \uplus \mathsf{dom}(w_{\mathsf{new}})$.

Proof. By induction on $\widehat{\rho} \vdash (e, \widehat{\sigma}, w) \Downarrow\!\Downarrow (\widehat{v}, \widehat{\sigma}', w', \psi)$. □

C.4 Correctness

Definition C.7. Every weight map w yields a distribution weight_w on $\mathsf{Model}_{\mathsf{dom}(w)}$:

$$\mathsf{weight}_w(m) = \prod_{(x \mapsto b) \in m} \text{if } b \text{ then } w(x) \text{ else } 1 - w(x)$$

Lemma C.8. $\rho, s \vdash (e, \sigma) \Downarrow (v, \sigma')$ iff $\mathsf{run}(\rho, e, \sigma)(s) = \langle \sigma' \setminus \sigma \rangle (v, \sigma')$.

Proof. The left-to-right direction follows by definition of run. For right-to-left direction, suppose $\rho, s \vdash (e, \sigma) \Downarrow (v_1, \sigma_1')$ witnesses $\mathsf{run}(\rho, e, \sigma)(s) = \langle \sigma_1' \backslash \sigma \rangle (v_1, \sigma_1')$. Then $\langle \sigma_1' \backslash \sigma \rangle (v_1, \sigma_1') = \langle \sigma' \backslash \sigma \rangle (v, \sigma')$ by Theorem C.2. □

Theorem C.9 (Correctness of Idealized Roulette). If $\widehat{\rho} \vdash (e, \widehat{\sigma}, w) \Downarrow\!\Downarrow (\widehat{v}, \widehat{\sigma}', w \uplus w', \alpha)$ and $\widehat{\rho}, \widehat{\sigma} \in \widehat{A}^{\mathsf{dom}(w)}$ and $\widehat{v}, \widehat{\sigma}', \alpha \in \widehat{A}^{\mathsf{dom}(w \uplus w')}$, then for all $m \in \mathsf{Model}_{\mathsf{dom}(w)}$ with $(\widehat{\rho}(m), e, \widehat{\sigma}(m)) \in \mathsf{Config}$,

$$\boxed{\widehat{\rho} \vdash (e, \widehat{\sigma}, w) \Downarrow (\widehat{v}, \widehat{\sigma}', w', \psi)}$$

Var
$$\frac{\widehat{\rho}(x) = [\varphi_i : v_i]_{i \in I}}{\widehat{\rho} \vdash (x, \widehat{\sigma}, w) \Downarrow ([\varphi_i : v_i]_{i \in I}, \widehat{\sigma}, w, \bigvee_i \varphi_i)}$$

Let
$$\frac{\widehat{\rho} \vdash (e_1, \widehat{\sigma}, w) \Downarrow (\widehat{v}_1, \widehat{\sigma}_1, w_1, \psi_1) \quad \widehat{\rho}[x \mapsto \widehat{v}_1] \vdash (e_2, \widehat{\sigma}_1, w_1) \Downarrow (\widehat{v}_2, \widehat{\sigma}_2, w_2, \psi_2)}{\widehat{\rho} \vdash (\texttt{let}\ x = e_1\ \texttt{in}\ e_2, \widehat{\sigma}, w) \Downarrow ([\psi_1 : \widehat{v}_2], [\psi_1 : \widehat{\sigma}_2], w_2, \psi_1 \wedge \psi_2)}$$

Lam
$$\frac{}{\widehat{\rho} \vdash (\lambda x.e, \widehat{\sigma}, w) \Downarrow ([\mathsf{T} : \texttt{clo}(\lambda x.e, \widehat{\rho})], \widehat{\sigma}, w, \mathsf{T})}$$

App
$$\frac{\widehat{\rho}(x_1) = [\varphi_i : \texttt{clo}(\lambda x_i.e_i, \widehat{\rho}_i)]_{1 \le i \in n} \uplus_{\mathsf{Closure}} \widehat{v} \quad \forall 1 \le i \le n.\ \widehat{\rho}_i[x_i \mapsto \widehat{\rho}(x_2)] \vdash (e_i, \widehat{\sigma}, w_{i-1}) \Downarrow (\widehat{v}_i, \widehat{\sigma}_i, w_i, \psi_i)}{\widehat{\rho} \vdash (x_1 x_2, \widehat{\sigma}, w_0) \Downarrow ([\varphi_i : \widehat{v}_i]_{i \in I}, [\varphi_i : \widehat{\sigma}_i]_{i \in I}, w_n, \bigvee_i (\varphi_i \wedge \psi_i))}$$

True
$$\frac{}{\widehat{\rho} \vdash (\texttt{true}, \widehat{\sigma}, w) \Downarrow ([\mathsf{T} : \texttt{true}], \widehat{\sigma}, w, \mathsf{T})}$$

False
$$\frac{}{\widehat{\rho} \vdash (\texttt{false}, \widehat{\sigma}, w) \Downarrow ([\mathsf{T} : \texttt{false}], \widehat{\sigma}, w, \mathsf{T})}$$

If
$$\frac{\widehat{\rho}(x) = [\varphi_1 : \texttt{true}, \varphi_2 : \texttt{false}] \uplus_{\mathsf{Bool}} \widehat{v} \quad \widehat{\rho} \vdash (e_1, \widehat{\sigma}, w) \Downarrow (\widehat{v}_1, \widehat{\sigma}_1, w_1, \psi_1) \quad \widehat{\rho} \vdash (e_2, \widehat{\sigma}, w_1) \Downarrow (\widehat{v}_2, \widehat{\sigma}_2, w_2, \psi_2)}{\widehat{\rho} \vdash (\texttt{if}\ x\ e_1\ e_2, \widehat{\sigma}, w) \Downarrow ([\varphi_1 : \widehat{v}_1, \varphi_2 : \widehat{v}_2], [\varphi_1 : \widehat{\sigma}_1, \varphi_2 : \widehat{\sigma}_2], w_2, (\varphi_1 \wedge \psi_1) \vee (\varphi_2 \wedge \psi_2))}$$

Num
$$\frac{r \in \mathbb{Q}}{\widehat{\rho} \vdash (r, \widehat{\sigma}, w) \Downarrow ([\mathsf{T} : r], \widehat{\sigma}, w, \mathsf{T})}$$

Arith
$$\frac{\widehat{\rho}(x_1) = [\varphi_i : r_i]_{i \in I} \uplus_{\mathsf{Num}} \widehat{v}_1 \quad \widehat{\rho}(x_2) = [\varphi_i : s_i]_{i \in I} \uplus_{\mathsf{Num}} \widehat{v}_2 \quad \forall\, i \in I.\ r_i, s_i \in \mathbb{Q} \quad \oplus \in \{+, -, \times\}}{\widehat{\rho} \vdash (x_1 \oplus x_2, \widehat{\sigma}, w) \Downarrow ([\varphi_i : r_i \mathbin{[\![\oplus]\!]} s_i]_{i \in I}, \widehat{\sigma}, w, \bigvee_i \varphi_i)}$$

Pair
$$\frac{\widehat{\rho}(x_1) = [\varphi_i : v_i]_{i \in I} \quad \widehat{\rho}(x_2) = [\varphi_i : w_i]_{i \in I}}{\widehat{\rho} \vdash ((x, y), \widehat{\sigma}, w) \Downarrow ([\varphi_i : (v_i, w_i)]_{i \in I}, \widehat{\sigma}, w, \bigvee_i \varphi_i)}$$

Fst
$$\frac{\widehat{\rho}(x) = [\varphi_i : (v_i, w_i)]_{i \in I} \uplus_{\mathsf{Pair}} \widehat{v}}{\widehat{\rho} \vdash (\texttt{fst}\, x, \widehat{\sigma}, w) \Downarrow ([\varphi_i : v_i]_{i \in I}, \widehat{\sigma}, w, \bigvee_{i \in I} \varphi_i)}$$

Snd
$$\frac{\widehat{\rho}(x) = [\varphi_i : (v_i, w_i)]_{i \in I} \uplus_{\mathsf{Pair}} \widehat{v}}{\widehat{\rho} \vdash (\texttt{snd}\, x, \widehat{\sigma}, w) \Downarrow ([\varphi_i : w_i]_{i \in I}, \widehat{\sigma}, w, \bigvee_{i \in I} \varphi_i)}$$

Ref
$$\frac{\widehat{\rho}(x) = [\varphi_i : v_i]_{i \in I} \quad \ell \text{ smallest not in } \mathrm{locs}(\widehat{\rho}, \widehat{\sigma})}{\widehat{\rho} \vdash (\texttt{ref}\, x, \widehat{\sigma}, w) \Downarrow ([\mathsf{T} : \ell], \widehat{\sigma}[\ell \mapsto [\varphi_i : v_i]_{i \in I}], w, \bigvee_i \varphi_i)}$$

Get
$$\frac{\widehat{\rho}(x) = [\varphi_i : \ell_i]_{i \in I} \uplus_{\mathsf{Loc}} \widehat{v} \quad \widehat{\sigma} = [\varphi_i : \sigma_i]_{i \in I} \uplus \widehat{\sigma}' \quad \forall i \in I.\ \ell_i \in \mathrm{dom}(\sigma_i)}{\widehat{\rho} \vdash (!x, \widehat{\sigma}, w) \Downarrow ([\varphi_i : \sigma_i(\ell_i)]_{i \in I}, \widehat{\sigma}, w, \bigvee_i \varphi_i)}$$

Set
$$\frac{\widehat{\rho}(x) = [\varphi_i : \ell_i]_{i \in I} \uplus_{\mathsf{Loc}} \widehat{v}_1 \quad \widehat{\rho}(y) = [\varphi_i : v_i]_{i \in I} \uplus_{\mathsf{Loc}} \widehat{v}_2 \quad \widehat{\sigma} = [\varphi_i : \sigma_i]_{i \in I} \uplus \widehat{\sigma}' \quad \forall i \in I.\ \ell_i \in \mathrm{dom}(\sigma_i)}{\widehat{\rho} \vdash (x := y, \widehat{\sigma}, w) \Downarrow ([\mathsf{T} : ()], [\varphi_i : \sigma_i[\ell_i \mapsto v_i]]_{i \in I}, w, \bigvee_i \varphi_i)}$$

Flip
$$\frac{\widehat{\rho}(x) = [\varphi_i : r_i]_{i \in I} \uplus_{\mathbb{Q}} \widehat{v} \quad \forall i \in I.\ s_i = \max(\min(r_i, 1), 0) \quad \{\alpha_i\}_{i \in I} \text{ smallest not in } \mathrm{symvars}(\widehat{\rho}, \widehat{\sigma}) \cup \mathrm{dom}(w)}{\widehat{\rho} \vdash (\texttt{flip}\, x, \widehat{\sigma}, w) \Downarrow ([\varphi_i : \alpha_i]_{i \in I}, \widehat{\sigma}, w \uplus \{\alpha_i \mapsto s_i\}_{i \in I}, \bigvee_i \varphi_i)}$$

Fail
$$\frac{}{\widehat{\rho} \vdash (\texttt{fail}, \widehat{\sigma}, w) \Downarrow (\varnothing, \varnothing, w, \bot)}$$

Fig. 18. Abstract semantics.

(1) The following random variables have the same distribution, where $[0,1]^{\mathbb{N}}$ is given the usual Lebesgue measure and $\mathsf{Model}_{\mathsf{dom}(w')}$ is given the measure $\mathsf{weight}_{w'}$:

$$\mathsf{run}(\widehat{\rho}(m), e, \widehat{\sigma}(m)) : [0,1]^{\mathbb{N}} \to \mathsf{Result}_\bot$$
$$\mathsf{result}_{\mathsf{dom}(w')}(\widehat{\sigma}(m), \widehat{v}|_m, \widehat{\sigma}'|_m) : \mathsf{Model}_{\mathsf{dom}(w')} \to \mathsf{Result}_\bot$$

(2) For all $m' \in \mathsf{Model}_{\mathsf{dom}(w')}$ it holds that $(\widehat{\rho}(m \uplus m'), \widehat{\sigma}(m \uplus m')) \neq \bot$ iff $\alpha(m \uplus m') = \mathsf{T}$.

Proof. Both (1) and (2) follow by induction on $\widehat{\rho} \vdash (e, \widehat{\sigma}, w) \Downarrow\!\Downarrow (\widehat{v}, \widehat{\sigma}', w \uplus w', \alpha)$. Point (2) is shown entirely analogously to Theorem B.21. For (1), we first consider the rules that do not involve probability:

- (Var) Suppose $\widehat{\rho}(x) = [\varphi_i : v_i]_{i \in I}$ and $\widehat{\rho} \vdash (x, \widehat{\sigma}, w) \Downarrow\!\Downarrow ([\varphi_i : v_i]_{i \in I}, \widehat{\sigma}, w, \bigvee_i \varphi_i)$. In this case $\mathsf{dom}(w') = \varnothing$ and $\mathsf{result}_\varnothing(\widehat{\sigma}(m), \widehat{\rho}(m)(x), \widehat{\sigma}(m))$ is the constant function taking value $\bot$ in case $x \notin \widehat{\rho}(m)$ and value v_i in case $\varphi_i(m) = \mathsf{T}$ for some $i \in I$; the same is true of $\mathsf{run}(\widehat{\rho}(m), x, \widehat{\sigma}(m))$, so they have the same distribution.
  Cases Lam, True, False, Num, Pair, Fail, Arith, Fst, Snd, Get, Set are similar.
- (Ref) Suppose
  - $\widehat{\rho}(x) = [\varphi_i : v_i]_{i \in I}$
  - ℓ smallest not in $\mathsf{locs}(\widehat{\rho}, \widehat{\sigma})$
  - $\widehat{\rho} \vdash (\mathtt{ref}\, x, \widehat{\sigma}, w) \Downarrow\!\Downarrow ([\mathsf{T} : \ell], \widehat{\sigma}[\ell \mapsto [\varphi_i : v_i]_{i \in I}], w, \bigvee_i \varphi_i)$

  In this case $\mathsf{dom}(w') = \varnothing$ and $\mathsf{result}_\varnothing(\widehat{\rho}(m), [\mathsf{T} : \ell]|_m, \widehat{\sigma}[\ell \mapsto [\varphi_i : v_i]_{i \in I}]|_m)$ is the constant function taking value $\bot$ if $x \notin \widehat{\rho}(m)$ and value $\langle \ell \rangle(\ell, \widehat{\sigma}(m)[\ell \mapsto v_i])$ if $\varphi_i(m) = \mathsf{T}$ for some $i \in I$. Similarly, $\mathsf{run}(\widehat{\rho}(m), \mathtt{ref}\, x, \widehat{\sigma}(m))$ is the constant function taking value $\bot$ if $x \notin \widehat{\rho}(m)$ and value $\langle \ell' \rangle(\ell', \widehat{\sigma}(m)[\ell' \mapsto v_i])$, if $\varphi_i(m) = \mathsf{T}$ for some $i \in I$, where ℓ' is an arbitrarily chosen fresh location. These functions are equal because $\langle \ell \rangle(\ell, \widehat{\sigma}(m)[\ell \mapsto v_i]) = \langle \ell' \rangle(\ell', \widehat{\sigma}(m)[\ell' \mapsto v_i])$, so they also have the same distribution.

For the rules involving probability,

- (If) Suppose
  - $\widehat{\rho}(x) = [\varphi_1 : \mathtt{true}, \varphi_2 : \mathtt{false}] \uplus_{\mathsf{Bool}} \widehat{v}$
  - $\widehat{\rho} \vdash (e_1, \widehat{\sigma}, w) \Downarrow\!\Downarrow (\widehat{v}_1, \widehat{\sigma}_1, w_1, \psi_1)$
  - $\widehat{\rho} \vdash (e_2, \widehat{\sigma}, w_1) \Downarrow\!\Downarrow (\widehat{v}_2, \widehat{\sigma}_2, w_2, \psi_2)$
  - $\widehat{\rho} \vdash (\mathtt{if}\, x\, e_1\, e_2, \widehat{\sigma}, w) \Downarrow\!\Downarrow ([\varphi_1 : \widehat{v}_1, \varphi_2 : \widehat{v}_2], [\varphi_1 : \widehat{\sigma}_1, \varphi_2 : \widehat{\sigma}_2], w_2, (\varphi_1 \wedge \psi_1) \vee (\varphi_2 \wedge \psi_2))$

  Let $X = \mathsf{result}_{w_1 \uplus w_2}(\widehat{\rho}(m), [\varphi_1 : \widehat{v}_1, \varphi_2 : \widehat{v}_2]|_m, [\varphi_1 : \widehat{\sigma}_1, \varphi_2 : \widehat{\sigma}_2]|_m)$, $Y = \mathsf{run}(\widehat{\rho}(m), \mathtt{if}\, x\, e_1\, e_2, \widehat{\sigma}(m))$. The goal is to show X and Y have the same distribution. There are three cases.
  - If $(\varphi_1 \vee \varphi_2)(m) = \bot$ then X and Y are both the constant function at $\bot$, and so have the same distribution.
  - If $\varphi_1(m) = \mathsf{T}$, then $X = \mathsf{result}_{w_1 \uplus w_2}(\widehat{\rho}(m), \widehat{v}_1|_m, \widehat{v}_2|_m)$ and $Y = \mathsf{run}(\widehat{\rho}(m), e_1, \widehat{\sigma}(m))$. Now $\mathsf{result}_{w_1 \uplus w_2}(\widehat{\rho}(m), \widehat{v}_1|_m, \widehat{\sigma}_1|_m)$ has the same distribution as $\mathsf{result}_{w_1}(\widehat{\rho}(m), \widehat{v}_1|_m, \widehat{\sigma}_1|_m)$ because $\mathsf{symvars}(\widehat{v}_1, \widehat{\sigma}_1) \subseteq \mathsf{dom}(w \uplus w_1)$, so the result follows from IH on e_1.
  - The case $\varphi_2(m) = \mathsf{T}$ is analogous, using the fact that $\mathsf{symvars}(\widehat{v}_2, \widehat{\sigma}_2) \subseteq \mathsf{dom}(w \uplus w_2)$.
- (App) Suppose
  - $\widehat{\rho}(x_1) = [\varphi_i : \mathtt{clo}(\lambda x_i.e_i, \widehat{\rho}_i)]_{1 \le i \in n} \uplus_{\mathsf{Closure}} \widehat{v}$
  - $\widehat{\rho}_i[x_i \mapsto \widehat{\rho}(x_2)] \vdash (e_i, \widehat{\sigma}, w_{i-1}) \Downarrow\!\Downarrow (\widehat{v}_i, \widehat{\sigma}_i, w_i, \psi_i)$ for all $1 \le i \le n$
  - $\widehat{\rho} \vdash (x_1 x_2, \widehat{\sigma}, w_0) \Downarrow\!\Downarrow ([\varphi_i : \widehat{v}_i]_{i \in I}, [\varphi_i : \widehat{\sigma}_i]_{i \in I}, w_n, \bigvee_i (\varphi_i \wedge \psi_i))$

  Let
  - $w' = w_0 \uplus \cdots \uplus w_n$
  - $X = \mathsf{result}_{w'}(\widehat{\sigma}(m), [\varphi_i : \widehat{v}_i]_{i \in I}|_m, [\varphi_i : \widehat{\sigma}_i]_{i \in I}|_m)$
  - $Y = \mathsf{run}(\widehat{\rho}(m), x_1 x_2, \widehat{\sigma}(m))$

with aim to show X and Y have the same distribution. There are two cases:
- If $\bigvee_{1 \le i \le n} \varphi_i(m) = \bot$ then X and Y are both constant functions at $\bot$, and so have the same distribution.
- If $\varphi_i(m) = \mathsf{T}$ for $1 \le i \le n$, then the following equations hold:

$$X = \mathsf{result}_{w'}(\widehat{\sigma}(m), \widehat{v}_i|_m, \widehat{\sigma}_i|_m)$$
$$Y = \mathsf{run}(\widehat{\rho}_i(m)[x_i \mapsto \widehat{\rho}(m)(x_2)], e_i, \widehat{\sigma}(m))$$

Since $\mathsf{symvars}(\widehat{\rho}_i, \widehat{\sigma}) \subseteq \mathsf{dom}(w)$, it holds that

$$Y = \mathsf{run}(\widehat{\rho}_i(m \uplus m_{1\ldots i-1})[x_i \mapsto \widehat{\rho}(m)(x_2)], e_i, \widehat{\sigma}(m \uplus m_{1\ldots i-1}))$$

for any $m_{1\ldots i-1} \in \mathsf{Model}_{\mathsf{dom}(w_1 \uplus \cdots \uplus w_{i-1})}$. Picking arbitrary such $m_{1\ldots i-1}$ gives, by IH, that Y has the same distribution as

$$X' = \mathsf{result}_{w \uplus w_1 \uplus \cdots \uplus w_{i-1}}(\widehat{\sigma}(m), \widehat{v}_i|_{m \uplus m_{1\ldots i-1}}, \widehat{\sigma}_i|_{m \uplus m_{1\ldots i-1}})$$

Now $\mathsf{symvars}(\widehat{v}_i, \widehat{\sigma}_i) \subseteq \mathsf{dom}(w) \uplus \mathsf{dom}(w_i)$ by choosing $\mathsf{dom}(w)$ for A in Theorem C.6, because $\mathsf{symvars}(\widehat{\rho}_i[x_i \mapsto \widehat{\rho}(x_2)], \widehat{\sigma}) \subseteq \mathsf{dom}(w)$. Thus

$$X' = \mathsf{result}_w(\widehat{\sigma}(m), \widehat{v}_i|_m, \widehat{\sigma}_i|_m) = X$$

as functions $\mathsf{Model}_{\mathsf{dom}(w_i)} \to \mathsf{Result}_\bot$, as needed.

- (Flip) Suppose
  - $\widehat{\rho}(x) = [\varphi_i : r_i]_{i \in I} \uplus_{[0,1]} \widehat{v}$
  - $s_i = \max(\min(r_i, 1), 0)$ for all $i \in I$
  - $\{\alpha_i\}_{i \in I}$ smallest not in $\mathsf{symvars}(\widehat{\rho}, \widehat{\sigma}) \cup \mathsf{dom}(w)$
  - $\widehat{\rho} \vdash (\mathtt{flip}\, x, \widehat{\sigma}, w) \Downarrow ([\varphi_i : \alpha_i]_{i \in I}, \widehat{\sigma}, w \uplus \{\alpha_i \mapsto s_i\}_{i \in I}, \bigvee_i \varphi_i)$

  and let
  - $\mu = \mathsf{weight}_{\{\alpha_i \mapsto s_i\}_{i \in I}}$
  - $X = \mathsf{result}_{\{\alpha_i\}_{i \in I}}(\widehat{\sigma}(m), [\varphi_i : \alpha_i]_{i \in I}|_m, \widehat{\sigma}(m)) = (m' \mapsto \langle \varnothing \rangle [\varphi_i : \alpha_i]_{i \in I}(m \uplus m'))$
  - $Y = \mathsf{run}(\widehat{\rho}(m), \mathtt{flip}\, x, \widehat{\sigma}(m))$.

  The goal is to show the distribution of X under μ is equal to the distribution of Y under Lebesgue measure on $[0,1]^{\mathbb{N}}$. There are two cases.
  - If $\bigvee_i \varphi_i(m) = \bot$, then both X and Y are constant functions at $\bot$, so they have the same distribution.
  - If $\varphi_i(m) = \mathsf{T}$ for some i, then X is the function defined by $X(m') = \langle \varnothing \rangle m'(\alpha_i)$ for all $m' \in \mathsf{Model}_{\{\alpha_i\}_{i \in I}}$, so has distribution $\mathsf{Ber}(s_i)$ under μ. Similarly, Y is the function

$$Y : [0,1]^{\mathbb{N}} \to \mathsf{Result}$$
$$Y(x :: s) = \begin{cases} \langle \varnothing \rangle \mathtt{true}, & \text{if } x < r_i \\ \langle \varnothing \rangle \mathtt{false}, & \text{if } x \ge r_i \end{cases}$$

  which also has distribution $\mathsf{Ber}(s_i)$ under Lebesgue measure on $[0,1]^{\mathbb{N}}$.
- (Let) Suppose
  - $\widehat{\rho} \vdash (e_1, \widehat{\sigma}, w) \Downarrow (\widehat{v}_1, \widehat{\sigma}_1, w \uplus w_1, \psi_1)$
  - $\widehat{\rho}[x \mapsto \widehat{v}_1] \vdash (e_2, \widehat{\sigma}_1, w \uplus w_1) \Downarrow (\widehat{v}_2, \widehat{\sigma}_2, w \uplus w_1 \uplus w_2, \psi_2)$.

  Fix arbitrary $r \in \mathsf{Result}$ with aim to show the following:

$$\Pr_{s \sim [0,1]^{\mathbb{N}}}\left[\mathsf{run}(\widehat{\rho}(m), \mathtt{let}\ x = e_1\ \mathtt{in}\ e_2, \widehat{\sigma}(m))(s) = r\right]$$
$$= \Pr_{\substack{m_1 \sim \mathsf{weight}_{w_1} \\ m_2 \sim \mathsf{weight}_{w_2}}}\left[\mathsf{result}_{\mathsf{dom}(w_1 \uplus w_2)}(\widehat{\sigma}(m), [\psi_1 : \widehat{v}_2]|_m, [\psi_1 : \widehat{\sigma}_2]|_m)(m_1 \uplus m_2) = r\right]$$

We will calculate from LHS to RHS. The calculation is long, but mostly due to explicit reasoning about renaming of heap locations; the structure of the proof mirrors that of the Let case of Theorem B.17. First, unwind the definition of run and perform inversion on the abstract semantics:

$$\text{LHS} = \Pr_{s_1,s_2\sim[0,1]^{\mathbb{N}}}\begin{bmatrix}\exists\, v_1\ \sigma_1\ v_2\ \sigma_2.\ \widehat{\rho}(m), s_1 \vdash (e_1, \widehat{\sigma}(m)) \Downarrow (v_1, \sigma_1) \wedge \\ \widehat{\rho}(m)[x \mapsto v_1], s_2 \vdash (e_2, \sigma_1) \Downarrow (v_2, \sigma_2) \wedge \\ \langle \sigma_2 \setminus \widehat{\sigma}(m)\rangle(v_2, \sigma_2) = r \wedge s \rightsquigarrow s_1, s_2\end{bmatrix}$$

Note that, as in the Let case of Theorem B.17, we have abused notation and written $\sigma_2 \setminus \widehat{\sigma}(m)$ instead of $\mathrm{dom}(\sigma_2) \setminus \mathrm{dom}(\widehat{\sigma}(m))$; we will continue to do this throughout for legibility's sake, automatically coercing stores into their domains as needed. Next, rewrite using Theorem C.8:

$$\cdots = \Pr_{s_1,s_2\sim[0,1]^{\mathbb{N}}}\begin{bmatrix}\exists\, v_1\ \sigma_1\ v_2\ \sigma_2. \\ \mathsf{run}(\widehat{\rho}(m), e_1, \widehat{\sigma}(m))(s_1) = \langle \sigma_1 \setminus \widehat{\sigma}(m)\rangle(v_1, \sigma_1) \wedge \\ \mathsf{run}(\widehat{\rho}(m)[x \mapsto v_1], e_2, \sigma_1)(s_2) = \langle \sigma_2 \setminus \sigma_1\rangle(v_2, \sigma_2) \wedge \\ \langle \sigma_2 \setminus \widehat{\sigma}(m)\rangle(v_2, \sigma_2) = r\end{bmatrix}$$

The event under consideration is a function of $\mathsf{run}(\widehat{\rho}(m), e_1, \widehat{\sigma}(m))(s_1)$, so the IH on e_1 applies:

$$\cdots = \Pr_{\substack{m_1\sim\mathsf{weight}_{w_1}\\ s_2\sim[0,1]^{\mathbb{N}}}}\begin{bmatrix}\exists\, v_1\ \sigma_1\ v_2\ \sigma_2. \\ \mathsf{result}_{\mathrm{dom}(w_1)}(\widehat{\sigma}(m), \widehat{v}_1|_m, \widehat{\sigma}_1|_m)(m_1) = \langle \sigma_1 \setminus \widehat{\sigma}(m)\rangle(v_1, \sigma_1) \wedge \\ \mathsf{run}(\widehat{\rho}(m)[x \mapsto v_1], e_2, \sigma_1)(s_2) = \langle \sigma_2 \setminus \sigma_1\rangle(v_2, \sigma_2) \wedge \\ \langle \sigma_2 \setminus \widehat{\sigma}(m)\rangle(v_2, \sigma_2) = r\end{bmatrix}$$

The equation $\mathsf{result}_{\mathrm{dom}(w_1)}(\widehat{\rho}(m), e_1, \widehat{\sigma}(m))(m_1) = \langle \sigma_1 \setminus \widehat{\sigma}(m)\rangle(v_1, \sigma_1)$ can be unpacked into a fresh set of locations F_1 and products of transpositions p_1, π_1 swapping $\mathrm{dom}(\widehat{\sigma}_1(m \uplus m_1)) \setminus \mathrm{dom}(\widehat{\sigma}(m))$ and $\mathrm{dom}(\sigma_1) \setminus \mathrm{dom}(\widehat{\sigma}(m))$ to F_1 respectively such that $p_1 \cdot (\widehat{v}_1(m \uplus m_1), \widehat{\sigma}_1(m \uplus m_1)) = \pi_1 \cdot (v_1, \sigma_1)$. Rearranging gives $p_1^{-1}\pi_1 \cdot (v_1, \sigma_1) = (\widehat{v}_1(m \uplus m_1), \widehat{\sigma}_1(m \uplus m_1))$, which combined with equivariance shows $\mathsf{run}(\widehat{\rho}(m)[x \mapsto v_1], e_2, \sigma_1)(s_2) = \langle \sigma_2 \setminus \sigma_1\rangle(v_2, \sigma_2)$ is equivalent to

$$\mathsf{run}(\widehat{\rho}(m)[x \mapsto \widehat{v}_1(m \uplus m_1)], e_2, \widehat{\sigma}_1(m \uplus m_1))(s_2) = p_1^{-1}\pi_1 \cdot \langle \sigma_2 \setminus \sigma_1\rangle(v_2, \sigma_2).$$

Thus we can continue as follows, where $\mathsf{Fresh}_1(F_1, p_1, \pi_1)$ is the logical formula expressing the side conditions on F_1, p_1, π_1 stated above:

$$\cdots = \Pr_{\substack{m_1\sim\mathsf{weight}_{w_1}\\ s_2\sim[0,1]^{\mathbb{N}}}}\begin{bmatrix}\exists\, v_1\ \sigma_1\ v_2\ \sigma_2\ F_1\ p_1\ \pi_1.\ \mathsf{Fresh}_1(F_1, p_1, \pi_1) \wedge \\ p_1 \cdot (\widehat{v}_1(m \uplus m_1), \widehat{\sigma}_1(m \uplus m_1)) = \pi_1 \cdot (v_1, \sigma_1) \wedge \\ \mathsf{run}(\widehat{\rho}(m)[x \mapsto \widehat{v}_1(m \uplus m_1)], e_2, \widehat{\sigma}_1(m \uplus m_1))(s_2) \\ \quad = p_1^{-1}\pi_1 \cdot \langle \sigma_2 \setminus \sigma_1\rangle(v_2, \sigma_2) \wedge \\ \langle \sigma_2 \setminus \widehat{\sigma}(m)\rangle(v_2, \sigma_2) = r\end{bmatrix}$$

After this rearrangement, the event under consideration is now a function of

$$(m_1, \mathsf{run}(\widehat{\rho}(m)[x \mapsto \widehat{v}_1(m \uplus m_1)], e_2, \widehat{\sigma}_1(m \uplus m_1))(s_2)),$$

so the IH on e_2 applies:

$$\cdots = \Pr_{\substack{m_1 \sim \mathsf{weight}_{w_1} \\ m_2 \sim \mathsf{weight}_{w_2}}} \left[\begin{array}{l} \exists\, v_1\ \sigma_1\ v_2\ \sigma_2\ F_1\ p_1\ \pi_1.\ \mathsf{Fresh}_1(F_1, p_1, \pi_1) \wedge \\ p_1 \cdot (\widehat{v}_1(m \uplus m_1), \widehat{\sigma}_1(m \uplus m_1)) = \pi_1 \cdot (v_1, \sigma_1) \wedge \\ \mathsf{result}_{\mathsf{dom}(w_2)}(\widehat{\sigma}_1(m \uplus m_1), \widehat{v}_2|_{m \uplus m_1}, \widehat{\sigma}_2|_{m \uplus m_1})(m_2) \\ \quad = p_1^{-1}\pi_1 \cdot \langle \sigma_2 \setminus \sigma_1 \rangle (v_2, \sigma_2) \wedge \\ \langle \sigma_2 \setminus \widehat{\sigma}(m) \rangle (v_2, \sigma_2) = r \end{array}\right]$$

The equation $\mathsf{result}_{\mathsf{dom}(w_2)}(\widehat{\sigma}_1(m \uplus m_1), \widehat{v}_2|_{m \uplus m_1}, \widehat{\sigma}_2|_{m \uplus m_1})(m_2) = p_1^{-1}\pi_1 \cdot \langle \sigma_2 \setminus \sigma_1 \rangle (v_2, \sigma_2)$ is equivalent to $\langle \widehat{\sigma}_2(m') \setminus \widehat{\sigma}_1(m') \rangle (\widehat{v}_2(m'), \widehat{\sigma}_2(m')) = p_1^{-1}\pi_1 \cdot \langle \sigma_2 \setminus \sigma_1 \rangle (v_2, \sigma_2)$, which rearranges to $\langle \widehat{\sigma}_2(m') \setminus \widehat{\sigma}_1(m') \rangle (p_1 \cdot (\widehat{v}_2(m'), \widehat{\sigma}_2(m'))) = \langle \sigma_2 \setminus \sigma_1 \rangle (\pi_1 \cdot (v_2, \sigma_2))$, where p_1, π_1 can be brought under $\langle - \rangle$ because the locations they touch are disjoint from $\widehat{\sigma}_2(m') \setminus \widehat{\sigma}_1(m')$ and $\sigma_2 \setminus \sigma_1$ respectively. This equation can then be further unpacked into a second fresh set of locations F_2 and products of transpositions p_2, π_2 swapping $\mathsf{dom}(\widehat{\sigma}_2(m')) \setminus \mathsf{dom}(\widehat{\sigma}_1(m'))$ and $\mathsf{dom}(\sigma_2) \setminus \mathsf{dom}(\sigma_1)$ with F_2 respectively such that $p_2 \cdot (p_1 \cdot (\widehat{v}_2(m'), \widehat{\sigma}_2(m'))) = \pi_2 \cdot (\pi_1 \cdot (v_2, \sigma_2))$. Since p_1, p_2 are disjoint products of transpositions, $p_2 p_1$ is itself a product of transpositions swapping $\mathsf{dom}(\widehat{\sigma}_2(m')) \setminus \mathsf{dom}(\widehat{\sigma}(m))$ with $F_1 \uplus F_2$; analogously, $\pi_2\pi_1$ swaps $\mathsf{dom}(\sigma_2) \setminus \mathsf{dom}(\widehat{\sigma}(m))$ with $F_1 \uplus F_2$. This implies $\langle \widehat{\sigma}_2(m') \setminus \widehat{\sigma}(m) \rangle (\widehat{v}_2(m'), \widehat{\sigma}_2(m')) = \langle \sigma_2 \setminus \widehat{\sigma}(m) \rangle (v_2, \sigma_2) = r$, so we can continue the calculation as follows, where $\mathsf{Fresh}_2(F_2, p_2, \pi_2)$ is the logical formula expressing the side conditions on F_2, p_2, π_2:

$$\cdots = \Pr_{\substack{m_1 \sim \mathsf{weight}_{w_1} \\ m_2 \sim \mathsf{weight}_{w_2}}} \underbrace{\left[\begin{array}{l} \exists\, v_1\ \sigma_1\ v_2\ \sigma_2\ F_1\ p_1\ \pi_1\ F_2\ p_2\ \pi_2. \\ \mathsf{Fresh}_1(F_1, p_1, \pi_1) \wedge \mathsf{Fresh}_2(F_2, p_2, \pi_2) \wedge \\ p_1 \cdot (\widehat{v}_1(m \uplus m_1), \widehat{\sigma}_1(m \uplus m_1)) = \pi_1 \cdot (v_1, \sigma_1) \wedge \\ \mathsf{let}\ m' = m \uplus m_1 \uplus m_2\ \mathsf{in} \\ p_2 p_1 \cdot (\widehat{v}_2(m'), \widehat{\sigma}_2(m')) = \pi_2\pi_1 \cdot (v_2, \sigma_2) \wedge \\ \langle \widehat{\sigma}_2(m') \setminus \widehat{\sigma}(m) \rangle (\widehat{v}_2(m'), \widehat{\sigma}_2(m')) = r \end{array}\right]}_{E(m_1, m_2)}$$

We are done if we can show that this is equal to the right-hand side

$$\Pr_{\substack{m_1 \sim \mathsf{weight}_{w_1} \\ m_2 \sim \mathsf{weight}_{w_2}}} \left[\mathsf{result}_{\mathsf{dom}(w_1 \uplus w_2)}(\widehat{\sigma}(m), [\psi_1 : \widehat{v}_2]|_m, [\psi_1 : \widehat{\sigma}_2]|_m)(m_1 \uplus m_2) = r\right].$$

Unwinding the definition of $\mathsf{result}_{\mathsf{dom}(w_1 \uplus w_2)}(-)$, this right-hand side is equivalent to

$$\Pr_{\substack{m_1 \sim \mathsf{weight}_{w_1} \\ m_2 \sim \mathsf{weight}_{w_2}}} \underbrace{\left[\mathsf{let}\ m' = m \uplus m_1 \uplus m_2\ \mathsf{in}\ \langle [\psi_1 : \widehat{\sigma}_2](m') \setminus \widehat{\sigma}(m) \rangle ([\psi_1 : \widehat{v}_2](m'), [\psi_1 : \widehat{\sigma}_2](m')) = r\right]}_{F(m_1, m_2)}.$$

It's enough to show E and F are equal as events.

($\subseteq$) First suppose $F(m_1, m_2)$, so $\langle [\psi_1 : \widehat{\sigma}_2](m') \setminus \widehat{\sigma}(m) \rangle ([\psi_1 : \widehat{v}_2](m'), [\psi_1 : \widehat{\sigma}_2](m')) = r$ where $m' = m \uplus m_1 \uplus m_2$. Then it must be that $\psi_1(m') = \mathsf{T}$ and $(\widehat{v}_2, \widehat{\sigma}_2)(m') \neq \bot$, and simplification gives $\langle \widehat{\sigma}_2(m') \setminus \widehat{\sigma}(m) \rangle (\widehat{v}_2(m'), \widehat{\sigma}_2(m')) = r$. Since $\psi_1(m') = \mathsf{T}$, IH on e_1 gives $(\widehat{v}_1(m'), \widehat{\sigma}_1(m')) \neq \bot$. Now we can always find disjoint sets of fresh variables F_1 and F_2 and swapping maps

π_1, p_1, π_2, p_2 for the pairs $(\widehat{v_1}(m'), \widehat{\sigma_1}(m'))$ and $(\widehat{v_2}(m'), \widehat{\sigma_2}(m'))$ so that setting

$$(v_1, \sigma_1) = \pi_1^{-1} p_1 \cdot (\widehat{v_1}(m \uplus m_1), \widehat{\sigma_1}(m \uplus m_2))$$
$$(v_2, \sigma_2) = \pi_1^{-1} \pi_2^{-1} p_2 p_1 \cdot (\widehat{v_2}(m'), \widehat{\sigma_2}(m'))$$

establishes $E(m_1, m_2)$.

($\supseteq$) Conversely, suppose $E(m_1, m_2)$ and let $m' = m \uplus m_1 \uplus m_2$. Since

$$(\widehat{v_1}(m'), \widehat{\sigma_1}(m')) = (\widehat{v_1}(m \uplus m_1), \widehat{\sigma_1}(m \uplus m')) \neq \bot,$$

IH on e_1 shows $\psi_1(m') = \mathsf{T}$. Thus

$$\langle [\psi_1 : \widehat{\sigma_2}](m') \setminus \widehat{\sigma}(m) \rangle ([\psi_1 : \widehat{v_2}](m'), [\psi_1 : \widehat{\sigma_2}](m'))$$
$$= \langle \widehat{v_2}(m') \setminus \widehat{\sigma}(m) \rangle (\widehat{v_1}(m'), \widehat{\sigma_2}(m')) \overset{\text{assumption}}{=} r$$

as needed to show $F(m_1, m_2)$. □

D Complete Performance Tables

D.1 Easy Benchmarks

Table 3. Easy Benchmarks

	Roulette			Dice (2025)			Dice (2020)		PSI
Benchmark	Ops	Size	Time (ms)	Ops	Size	Time (ms)	Size	Time (ms)	Time (ms)
ALARM	32	11	0	55	11	19	11	16	71
EVIDENCE1	6	3	0	15	5	24	5	17	22
EVIDENCE2	12	4	0	17	6	21	6	18	34
GRASS	40	13	0	72	15	30	15	19	80
MURDER-MYSTERY	6	4	0	297	6	28	6	20	46
NOISY-OR	143	33	0	170	35	23	35	19	328
TWO-COINS	6	3	0	15	5	25	5	22	20

D.2 Expressive Benchmarks

Table 4. Network Reliability

	Roulette			Enumeration
Iterations	Operations	Size	Time	Time
1	344	14	1	0
2	3,181	95	3	3
3	12,692	283	12	164
4	31,765	547	15	3,719
5	75,258	821	19	–
10	149,541	2,191	38	–
50	1,607,021	13,151	421	–
100	5,903,871	26,851	1,527	–
200	22,747,571	54,251	6,267	–
400	89,434,971	109,051	26,569	–

Table 5. Hardware N

N	Roulette Ops	Roulette Size	Roulette Time (ms)	Enumeration Time (ms)
1	977	23	3	1,114
2	816	31	2	1,024
3	752	24	3	823
4	752	24	3	836
5	752	24	3	818
6	752	24	3	824
7	752	24	3	819
8	752	24	3	823
9	752	24	3	824
10	752	24	3	831

Table 6. Hardware Fuel

Fuel	Roulette Ops	Roulette Size	Roulette Time (ms)	Enumeration Time (ms)
1	0	0	0	0
3	2,176	70	4	1,102
5	7,806	218	7	–
7	18,710	492	12	–
9	35,210	780	16	–
11	57,218	1,068	20	–
21	251,174	2,508	60	–
31	584,558	3,948	129	–
41	1,057,514	5,388	226	–
51	1,669,898	6,828	358	–
61	2,421,854	8,268	560	–
71	3,313,238	9,708	703	–
81	4,344,194	11,148	929	–
91	5,514,578	12,588	1,235	–
93	5,765,438	12,876	1,264	–
95	6,021,806	13,164	1,299	–
97	6,283,826	13,452	1,361	–
99	6,551,354	13,740	1,406	–

Table 7. Hardware Bitwidth

	Roulette			Enumeration
Bitwidth	Ops	Size	Time (ms)	Time (ms)
5	1,593	24	4	1,064
6	1,593	24	4	983
7	1,593	24	4	1,014
8	1,593	24	4	960
9	1,593	24	4	952
10	1,593	24	4	897
11	1,593	24	4	931
12	1,593	24	4	866
13	1,593	24	4	864
14	1,593	24	4	835
15	1,593	24	4	852
16	1,593	24	4	815
17	1,593	24	5	825
18	1,593	24	5	795
19	1,593	24	5	806
20	1,593	24	5	794

D.3 Scaling Benchmarks

Table 8. Caesar with Errors

	Roulette			Dice (2025)			Dice (2020)	
Iterations	Ops	Size	Time (ms)	Ops	Size	Time (ms)	Size	Time (ms)
1	5,074	259	13	521,687	195	75	751	28
26	109,397	4,953	124	12,908,645	5,959	1,810	13,238	123
51	214,338	9,776	221	25,280,143	11,775	3,588	25,767	215
76	319,979	14,851	337	37,648,765	17,651	5,414	38,942	399
101	425,262	19,752	432	50,020,149	23,352	7,215	51,769	612
126	528,393	24,135	582	62,381,755	29,213	8,942	63,686	680
151	634,084	29,431	686	74,752,953	35,039	10,769	76,009	715
176	740,049	34,737	796	87,111,831	41,033	12,856	89,118	878
201	844,128	39,164	890	99,481,979	46,824	14,849	102,078	1,101
226	950,397	44,580	1,015	111,843,093	52,790	16,853	114,871	1,198
251	1,054,452	49,144	1,164	124,213,559	58,520	18,715	127,122	1,085
276	1,158,379	53,777	1,278	136,592,621	64,321	21,007	139,493	1,214

Table 9. Diamond Network

	Roulette			Dice (2025)			Dice (2020)	
Iterations	Ops	Size	Time (ms)	Ops	Size	Time (ms)	Size	Time (ms)
1	6	2	0	96	4	39	4	14
100	1,887	200	16	26,737	202	35	202	13
200	3,787	400	28	93,441	402	56	402	19
300	5,687	600	39	200,141	602	81	602	23
400	7,587	800	54	346,841	802	122	802	22
500	9,487	1,000	67	533,541	1,002	165	1,002	23
700	13,287	1,400	88	1,026,941	1,402	281	1,402	26
800	15,187	1,600	117	1,333,641	1,602	360	1,602	27
900	17,087	1,800	130	1,680,341	1,802	435	1,802	28
1,000	18,987	2,000	139	2,067,041	2,002	508	2,002	38
2,000	37,987	4,000	286	8,134,047	4,002	2,004	4,002	43
3,000	56,987	6,000	425	18,201,047	6,002	4,808	6,002	68
4,000	75,987	8,000	612	32,268,049	8,002	9,162	8,002	82
5,000	94,987	10,000	747	50,335,051	10,002	15,557	10,002	103

Table 10. Ladder Network

	Roulette			Dice (2025)			Dice (2020)	
Iterations	Ops	Size	Time (ms)	Ops	Size	Time (ms)	Size	Time (ms)
1	14	3	0	313	6	16	6	11
5	173	19	7	2,145	30	22	30	12
10	368	39	8	7,968	60	23	60	18
20	758	79	9	2,116,168	120	55	120	12
100	3,878	399	20	–	–	–	600	14
200	7,778	799	33	–	–	–	1,200	24
300	11,678	1,199	50	–	–	–	1,800	18
400	15,578	1,599	63	–	–	–	2,400	21
500	19,478	1,999	87	–	–	–	3,000	26
700	27,278	2,799	122	–	–	–	4,200	37
800	31,178	3,199	135	–	–	–	4,800	33
900	35,078	3,599	150	–	–	–	5,400	41
1,000	38,978	3,999	168	–	–	–	6,000	44
2,000	77,978	7,999	339	–	–	–	12,000	83
3,000	116,978	11,999	552	–	–	–	18,000	123
4,000	155,978	15,999	731	–	–	–	24,000	158
5,000	194,978	19,999	909	–	–	–	30,000	208

Table 11. Figure 1 of Holtzen et al. [29]

	Roulette			Dice (2025)			Dice (2020)	
Iterations	Ops	Size	Time (ms)	Ops	Size	Time (ms)	Size	Time (ms)
1	11	4	0	91	5	17	5	14
100	1,100	301	10	67,806	203	44	203	19
200	2,200	601	14	255,768	403	75	403	18
300	3,300	901	19	564,126	603	131	603	20
400	4,400	1,201	23	992,834	803	206	803	17
500	5,500	1,501	28	1,541,508	1,003	300	1,003	22
700	7,700	2,101	37	2,998,680	1,403	557	1,403	26
800	8,800	2,401	42	3,907,472	1,603	728	1,603	29
900	9,900	2,701	49	4,936,410	1,803	898	1,803	30
1,000	11,000	3,001	50	6,085,382	2,003	1,101	2,003	31
2,000	22,000	6,001	118	24,175,120	4,003	4,661	4,003	47
3,000	33,000	9,001	178	54,270,994	6,003	11,825	6,003	71
4,000	44,000	12,001	230	96,364,856	8,003	23,738	8,003	83
5,000	55,000	15,001	287	–	–	–	10,003	101

D.4 Hidden Markov Model

Table 12. Hidden Markov Models for MARGINAL-LIKELIHOOD Query

	Roulette			Dice (2025)			Dice (2020)		DP
Steps	Ops	Size	Time (ms)	Ops	Size	Time (ms)	Size	Time (ms)	Time (ms)
1	10	5	0	354	7	1,266	7	505	0
100	61,093	401	29	191,918	403	1,359	403	468	16
500	1,505,493	2,001	391	4,160,320	2,003	2,259	2,003	533	45
1,000	6,010,993	4,001	1,585	16,321,614	4,003	5,168	4,003	746	81
2,000	24,021,993	8,001	6,905	–	–	–	8,003	1,608	167
3,000	54,032,993	12,001	15,377	–	–	–	12,003	2,983	238
4,000	–	–	–	–	–	–	16,003	5,315	314
5,000	–	–	–	–	–	–	20,003	8,561	397

Table 13. Hidden Markov Models for FILTERING-MARGINAL Query

	Roulette			Dice (2025)			Dice (2020)		DP
Steps	Ops	Size	Time (ms)	Ops	Size	Time (ms)	Size	Time (ms)	Time (ms)
1	24	11	0	368	10	1,270	10	472	0
100	82,095	1,199	31	174,112	604	1,376	604	465	16
500	2,010,495	5,999	524	3,671,314	3,004	2,177	3,004	520	44
1,000	8,020,995	11,999	2,191	14,343,608	6,004	4,755	6,004	671	79
2,000	32,041,995	23,999	9,190	–	–	–	12,004	1,304	164
3,000	72,062,995	35,999	22,041	–	–	–	18,004	2,355	235
4,000	–	–	–	–	–	–	24,004	3,848	313
5,000	–	–	–	–	–	–	30,004	5,559	399

Table 14. Hidden Markov Models for SMOOTHING-MARGINAL Query

	Roulette			Dice (2025)			Dice (2020)		DP
Steps	Ops	Size	Time (ms)	Ops	Size	Time (ms)	Size	Time (ms)	Time (ms)
1	16	7	0	360	8	1,274	8	463	0
100	80,503	403	29	172,520	404	1,314	404	475	16
500	2,002,503	2,003	525	3,663,322	2,004	2,180	2,004	528	44
1,000	8,005,003	4,003	2,198	14,327,616	4,004	4,773	4,004	677	80
2,000	32,010,003	8,003	9,446	–	–	–	8,004	1,217	165
3,000	72,015,003	12,003	21,331	–	–	–	12,004	2,351	243
4,000	–	–	–	–	–	–	16,004	3,675	317
5,000	–	–	–	–	–	–	20,004	5,760	394

Table 15. Hidden Markov Models for FILTERING-JOINT Query

	Roulette			Dice (2025)			Dice (2020)		DP
Steps	Ops	Size	Time (ms)	Ops	Size	Time (ms)	Size	Time (ms)	Time (ms)
1	23	9	0	368	9	1,269	9	459	0
100	96,745	999	34	186,362	554	1,311	554	478	19
500	2,383,745	4,999	581	3,982,564	2,754	2,198	2,754	549	60
1,000	9,517,495	9,999	2,465	15,591,108	5,504	5,075	5,504	712	113
2,000	38,034,995	19,999	10,911	–	–	–	11,004	1,651	237
3,000	85,552,495	29,999	25,219	–	–	–	16,504	3,112	342
4,000	–	–	–	–	–	–	22,004	5,251	475
5,000	–	–	–	–	–	–	27,504	8,107	580

Table 16. Hidden Markov Models for SMOOTHING-JOINT Query

	Roulette			Dice (2025)			Dice (2020)		DP
Steps	Ops	Size	Time (ms)	Ops	Size	Time (ms)	Size	Time (ms)	Time (ms)
1	10	5	0	354	7	1,289	7	457	0
100	85,742	597	30	175,456	453	1,336	453	476	17
500	2,128,742	2,997	547	3,728,058	2,253	2,159	2,253	551	51
1,000	8,507,492	5,997	2,295	14,582,102	4,503	4,751	4,503	736	94
2,000	34,014,992	11,997	9,641	–	–	–	9,003	1,606	199
3,000	76,522,492	17,997	23,391	–	–	–	13,503	3,234	288
4,000	–	–	–	–	–	–	18,003	5,676	392
5,000	–	–	–	–	–	–	22,503	8,796	491

D.5 Bayesian Networks

Table 17. Bayesian Networks

	Roulette			Dice (2025)			Dice (2020)	
Benchmark	Ops	Size	Time (ms)	Ops	Size	Time (ms)	Size	Time (ms)
CANCER	43	13	2	149	15	49	28	19
SURVEY	256	46	4	708	48	29	73	18
ALARM	5,007	981	42	1,496,332	672	308	1,366	30
INSURANCE	1,855,308	75,594	395	2,056,519	44,846	643	101,047	148
HEPAR2	77,362	1,967	140	380,225	1,969	230	3,936	32
HAILFINDER	2,877,324	33,211	596	–	–	–	65,386	428
PIGS	106	19	255	822,635	25	417	35	48
WATER	188,397	39,146	200	584,307	33,226	454	51,952	16,083
MUNIN	193,689	10,307	2,400	337,439,533	3,704	24,839	11,977	1,605

E Diamond Two Ways

This section shows two different ways in which the diamond network from our network reliability example in Section 2.1 can be implemented.

The following code realizes the diamond network with linear scaling, using a non-tail recursive function to perform every `flip` and then perform Boolean operations back to front:

```
(define (main n)
  (let go ([n n])
    (cond
      [(zero? n) #t]
      [else
       (define drop (flip 0.0001))
       (define route (flip 0.5))
       (define s1 (go (sub1 n)))
       (define s2 (if route s1 #f))
       (define s3 (if route #f s1))
       (or s2 (and s3 (not drop)))])))
```

The following code realizes the diamond network with quadratic scaling, using a tail-recursive function:

```
(define (main n)
  (let go ([n n] [s1 #t])
    (cond
      [(zero? n) s1]
      [else
       (define drop (flip 0.0001))
       (define route (flip 0.5))
       (define s2 (if route s1 #f))
       (define s3 (if route #f s1))
       (go (sub1 n) (or s2 (and s3 (not drop))))])))
```

Received 2024-11-15; accepted 2025-03-06