Graph IRs for Impure Higher-Order Languages (Technical Report)

OLIVER BRAČEVAC, Purdue University, USA
GUANNAN WEI, Purdue University, USA
SONGLIN JIA, Purdue University, USA
SUPUN ABEYSINGHE, Purdue University, USA
YUXUAN JIANG, Purdue University, USA
YUYAN BAO, Augusta University, USA
TIARK ROMPF, Purdue University, USA

This is a companion report for the OOPSLA 2023 paper of the same title, presenting a detailed end-to-end account of the λ^*_{G} graph IR, at a level of detail beyond a regular conference paper. Our first concern is adequacy and soundness of λ^*_{G}, which we derive from a direct-style imperative functional language (a variant of Bao et al.'s λ^*-calculus with reachability types and a simple effect system) by a series of type-preserving translations into a calculus in monadic normalform (MNF). Static reachability types and effects entirely inform λ^*_{G}'s dependency synthesis. We argue for its adequacy by proving its functional properties along with *dependency safety* via progress and preservation lemmas with respect to a notion of call-by-value (CBV) reduction that checks the observed order of effects.

Our second concern is establishing the correctness of λ^*_{G}'s equational rules that drive compiler optimizations (*e.g.*, DCE, λ-hoisting, etc.), by proving contextual equivalence using logical relations. A key insight is that the functional properties of dependency synthesis permit a logical relation on λ^*_{G} in MNF in terms of previously developed logical relations for the direct-style λ^*-calculus.

Finally, we also include a longer version of the conference paper's section on code generation and code motion for λ^*_{G} as implemented in Scala LMS.

Contents

List of Figures

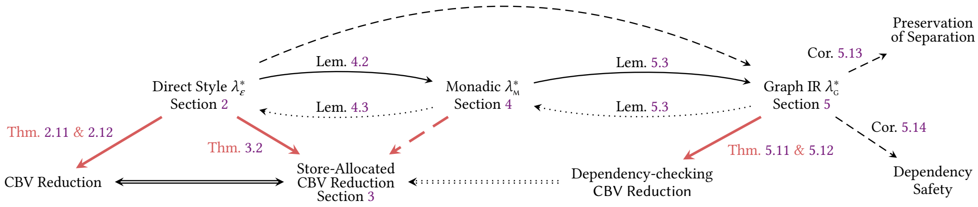

Fig. 1. Overview of the calculi and their metatheory in this report. Thick red arrows indicate type soundness proofs w.r.t. a reduction semantics. Black arrows indicate type preservation proofs between calculi. Dashed arrows indicate corollaries, and dotted arrows indicate an embedding or erasure. Double arrows indicate an equivalence.

1 INTRODUCTION

This document complements the paper “Graph IRs for Impure Higher-Order Languages” [Bračevac et al. 2023] and presents a detailed end-to-end account of the λ^*_{G} graph IR, step by step refinining a series of calculi and the formal development of their metatheory, in a way that goes beyond the space available in a conference paper. The process begins from the direct-style λ^*_ε-calculus (a variant of Bao et al. [2021]’s λ^*-calculus with a simple effect system), proceeds with λ^*_{M}, a version in monadic normal form (MNF), and ends in the λ^*_{G}-calculus, the typed graph IR in monadic normal form (MNF)[1] with effect dependencies. Each stage constitutes a straightforward refinement which is provably type/effect/qualifier preserving, and is provably type-safe with respect to a notion of call-by-value (CBV) reduction, established by standard progress and preservation lemmas.

The key end-to-end safety guarantees are visualized in Figure 1. The result is a sequence of corollaries establishing key properties, namely that the direct-style language can be translated in a type-preserving and dependency-synthesizing manner into λ^*_{G}, and that λ^*_{G} is sound with respect to a dependency-checking call-by-value operational semantics. It follows that synthesized dependencies correctly reflect the execution order of effects, and Bao et al. [2021]’s preservation of separation holds for the graph IR, a memory property guaranteeing that reductions never cause disjoint graph IR computations to become aliased.

Leveraging the type-and-effects safety in the λ^*_ε-calculus, the equational rules shown in the paper (Section 5.1) are proved sound by contextual equivalence via logical relations, building upon a framework developed in parallel with this report [Bao et al. 2023].

[1] Monadic normal form [Hatcliff and Danvy 1994] is a generalization of ANF [Flanagan et al. 1993] and related let-normal forms, where let bindings permit nesting.

We present the corresponding development in detail as follows:

- **The direct-style λ^*_ε-calculus (Section 2):** we introduce the syntax and typing rules of our base calculus with reachability types and effects, motivating the basic design principles and features of the system. The formalization and proofs very closely follow the publicly available Coq mechanizations of the original λ^*-calculus[2]. λ^*_ε as presented in this report lacks some features of the original λ^*-calculus (e.g., no recursion, no escaping closures, flat mutable references) which are non-essential to understand the core ideas.
- **The λ^*_ε-calculus with store-allocated values (Section 3):** as a stepping stone towards monadic normal form, we refine the call-by-value operational semantics of λ^*_ε to place both mutable references and immutable introduction forms in the store, and prove the direct-style type system sound w.r.t. this refined semantics. Notably, substitutions become simpler, because they are just renamings of variables to store locations.
- **The monadic normal form λ^*_{M} (Section 4):** we define a provably type/effect/qualifier-preserving translation of the direct-style λ^*_ε-calculus into λ^*_{M} which is in monadic normal form (MNF). This normal form generalizes the previously used A-normal form (ANF) by permitting nested terms. Unlike ANF, reductions preserve the monadic normalform at all times, even under β-reduction. We further establish that λ^*_{M} is a proper sublanguage of λ^*_ε, by (1) proving that MNF terms can always be assigned the same type, effect, and reachability qualifiers in both systems, and (2) proving that reduction with store-allocated values (Section 3) preserves MNF. Type soundness of λ^*_{M} follows from type soundness of λ^*_ε as a corollary.
- **The graph IR λ^*_{G} with (hard) dependencies (Section 5):** we enrich the MNF-calculus λ^*_{M} with effect dependencies in the λ^*_{G}-calculus. Dependencies are entirely determined by reachability qualifiers and effects. We prove type soundness and preservation of separation with respect to a stricter operational semantics, which establishes *dependency safety*: evaluation respects the order of effect dependencies for well-typed graph IR terms, *i.e.*, an effectful graph node is executed only if all its dependencies have already been resolved. We also prove end-to-end type/effect/qualifier preservation and effect synthesis from the direct-style λ^*_ε-calculus into the λ^*_{G} graph IR.
- **The graph IR λ^*_{G} in MNF with hard and soft dependencies (Section 6):** we refine the effects of λ^*_{G} from mere uses to a read and write distinction, which synthesize into hard and soft dependencies.
- **The equational theory of λ^*_ε (Section 7):** We develop logical relations over reachability types and effects, which enables reasoning about contextual equivalence of λ^*_ε terms.
- **Optimization rules and equational theory of λ^*_{G} (Section 8):** We prove soundness of the optimization rules in the main paper for the λ^*_{G} graph IR with hard dependencies in terms of contextual equivalence. We leverage the results from Sections 4, 6 and 7, to derive the logical relations argument by appealing to the direct-style system through a "round-trip" translation erasing and re-synthesizing hard dependencies. We leave the logical relations argument for the system including soft dependencies as future work.
- **Code motion algorithms for λ^*_{G} (Section 9):** we present the code motion algorithms that transform graphs into trees and emit code. A vanilla code motion algorithm is presented first, which is then extended with frequency estimation and compact code generation.

[2] http://github.com/tiarkrompf/reachability

Syntax $\boxed{\lambda^*_\varepsilon}$

$$
\begin{array}{llll}
x, y, z & \in & \mathsf{Var} & \text{Variables} \\
\ell, w & \in & \mathsf{Loc} & \text{Locations} \\
v & ::= & c \mid \lambda x.t \mid \ell & \text{Values} \\
t & ::= & v \mid \ell \mid x \mid t\ t \mid \mathbf{ref}_\ell\ t \mid !\,t \mid t := t \mid \mathbf{let}\ x = t\ \mathbf{in}\ t & \text{Terms} \\
p, q, r, \varepsilon, \varphi & \in & \mathcal{P}_{\mathsf{fin}}(\mathsf{Var} \uplus \mathsf{Loc}) & \text{Qualifiers/Effects/Observations} \\
S, T, U, V & ::= & B \mid (x : T^q) \to^{\varepsilon} T^q \mid \mathsf{Ref}\ T & \text{Types} \\
\Gamma & ::= & \varnothing \mid \Gamma, x : T^q & \text{Typing Environments} \\
\Sigma & ::= & \varnothing \mid \Sigma, \ell : T^q & \text{Store Typing}
\end{array}
$$

Term Typing $\boxed{[\Sigma \mid \Gamma]^{\varphi} \vdash t : T^q\ \varepsilon}$

$$\frac{c \in B}{[\Sigma \mid \Gamma]^{\varphi} \vdash c : B^{\varnothing}\ \varnothing} \quad (\text{T-CST})$$

$$\frac{[\Sigma \mid \Gamma]^{\varphi} \vdash t_1 : S^p\ \varepsilon_1 \qquad [\Sigma \mid \Gamma, x : S^{p* \cap \varphi*}]^{\varphi, x} \vdash t_2 : T^q\ \varepsilon_2 \qquad \theta = [p/x] \quad x \notin \mathrm{fv}\,(T)}{[\Sigma \mid \Gamma]^{\varphi} \vdash \mathbf{let}\ x = t_1\ \mathbf{in}\ t_2 : (T^q\ \varepsilon_1 \rhd \varepsilon_2)\theta} \quad (\text{T-LET})$$

$$\frac{x : T^q \in \Gamma \qquad x \subseteq \varphi}{[\Sigma \mid \Gamma]^{\varphi} \vdash x : T^x\ \varnothing} \quad (\text{T-VAR})$$

$$\frac{[\Sigma \mid \Gamma]^{\varphi} \vdash t_1 : \mathsf{Alloc}^q\ \varepsilon_1 \qquad [\Sigma \mid \Gamma]^{\varphi} \vdash t_2 : B^{\varnothing}\ \varepsilon_2}{[\Sigma \mid \Gamma]^{\varphi} \vdash \mathbf{ref}_{t_1}\ t_2 : (\mathsf{Ref}\ B)^{\varnothing}\ \varepsilon_1 \rhd \varepsilon_2 \rhd q} \quad (\text{T-REF})$$

$$\frac{\ell : T^q \in \Sigma \qquad \ell \subseteq \varphi}{[\Sigma \mid \Gamma]^{\varphi} \vdash \ell : T^{\ell}\ \varnothing} \quad (\text{T-LOC})$$

$$\frac{[\Sigma \mid \Gamma]^{\varphi} \vdash t : (\mathsf{Ref}\ B)^q\ \varepsilon}{[\Sigma \mid \Gamma]^{\varphi} \vdash !t : B^{\varnothing}\ \varepsilon \rhd q} \quad (\text{T-!})$$

$$\frac{[\Sigma \mid \Gamma, x : T^p]^{q, x} \vdash t : U^r\ \varepsilon \qquad q \subseteq \varphi}{[\Sigma \mid \Gamma]^{\varphi} \vdash \lambda x.t : (x : T^p \to^{\varepsilon} U^r)^q\ \varnothing} \quad (\text{T-ABS})$$

$$\frac{[\Sigma \mid \Gamma]^{\varphi} \vdash t_1 : (\mathsf{Ref}\ B)^q\ \varepsilon_1 \qquad [\Sigma \mid \Gamma]^{\varphi} \vdash t_2 : B^{\varnothing}\ \varepsilon_2}{[\Sigma \mid \Gamma]^{\varphi} \vdash t_1 := t_2 : \mathsf{Unit}^{\varnothing}\ \varepsilon_1 \rhd \varepsilon_2 \rhd q} \quad (\text{T-:=})$$

$$\frac{\begin{array}{c}[\Sigma \mid \Gamma]^{\varphi} \vdash t_1 : \left(x : T^{p* \cap q*} \to^{\varepsilon_3} U^r\right)^q\ \varepsilon_1 \\ [\Sigma \mid \Gamma]^{\varphi} \vdash t_2 : T^p\ \varepsilon_2 \qquad \theta = [p/x] \\ x \notin \mathrm{fv}(U) \qquad \varepsilon_3 \subseteq q, x \qquad r \subseteq \varphi, x\end{array}}{[\Sigma \mid \Gamma]^{\varphi} \vdash t_1\ t_2 : (U^r\ \varepsilon_1 \rhd \varepsilon_2 \rhd \varepsilon_3)\theta} \quad (\text{T-APP})$$

$$\frac{[\Sigma \mid \Gamma]^{\varphi} \vdash t : S^p\ \varepsilon_1 \qquad \Sigma \mid \Gamma \vdash S^p\ \varepsilon_1 <: T^q\ \varepsilon_2 \qquad q, \varepsilon_2 \subseteq \varphi}{[\Sigma \mid \Gamma]^{\varphi} \vdash t : T^q\ \varepsilon_2} \quad (\text{T-SUB})$$

Subtyping $\boxed{\Sigma \mid \Gamma \vdash q <: q}$ $\boxed{\Sigma \mid \Gamma \vdash T <: T}$ $\boxed{\Sigma \mid \Gamma \vdash T^q\ \varepsilon <: T^q\ \varepsilon}$

$$\frac{p \subseteq q \subseteq dom(\Gamma) \cup dom(\Sigma)}{\Sigma \mid \Gamma \vdash p <: q} \quad (\text{Q-SUB})$$

$$\frac{\Sigma \mid \Gamma \vdash U^q\ \varnothing <: S^o\ \varnothing \qquad \Gamma, x : U^p \mid \Sigma \vdash T^q\ \varepsilon_1 <: V^r\ \varepsilon_2}{\Sigma \mid \Gamma \vdash (x : S^o) \to^{\varepsilon_1} T^q <: (x : U^p) \to^{\varepsilon_2} V^r} \quad (\text{S-FUN})$$

$$\frac{}{\Sigma \mid \Gamma \vdash B <: B} \quad (\text{S-BASE})$$

$$\frac{\Sigma \mid \Gamma \vdash S <: T \qquad \Sigma \mid \Gamma \vdash p <: q \qquad \Sigma \mid \Gamma \vdash \varepsilon_1 <: \varepsilon_2}{\Sigma \mid \Gamma \vdash S^p\ \varepsilon_1 <: T^q\ \varepsilon_2} \quad (\text{SQE-SUB})$$

$$\frac{}{\Sigma \mid \Gamma \vdash \mathsf{Ref}\ B <: \mathsf{Ref}\ B} \quad (\text{S-REF})$$

Fig. 2. The direct-style λ^*_ε-calculus.

2 THE DIRECT-STYLE λ^*_ε-CALCULUS

2.1 Overview

The λ^*_ε-calculus is a variant of Bao et al.'s λ^*-calculus. The original system features an effect system based on Gordon [2021]'s effect quantale framework. For simplicity, we only consider a

Qualifier Substitution $\boxed{q[x \mapsto p]}$

$$
\begin{array}{ll}
q[p/x] = q \setminus \{x\} \cup p & x \in q \\
q[p/x] = q & x \notin q
\end{array}
$$

Reachability $\boxed{\Sigma \mid \Gamma \vdash x \rightsquigarrow x}$ $\boxed{\Sigma \mid \Gamma \vdash \ell \rightsquigarrow \ell}$ $\boxed{\Sigma \mid \Gamma \vdash q*}$

$$
\begin{array}{llll}
\Sigma \mid \Gamma \vdash x \rightsquigarrow y & \Leftrightarrow & x : T^{q,y} \in \Gamma & \text{Reachability Relation (Variables)} \\
\Sigma \mid \Gamma \vdash \ell \rightsquigarrow w & \Leftrightarrow & \ell : T^{q,w} \in \Sigma & \text{Reachability Relation (Locations)} \\
\Sigma \mid \Gamma \vdash x* & := & \{\, y \mid x \rightsquigarrow^* y \,\} & \text{Variable Saturation} \\
\Sigma \mid \Gamma \vdash \ell* & := & \{\, w \mid \ell \rightsquigarrow^* w \,\} & \text{Location Saturation} \\
\Sigma \mid \Gamma \vdash q* & := & \bigcup_{x \in q} x* \cup \bigcup_{\ell \in q} \ell* & \text{Qualifier Saturation}
\end{array}
$$

Effects $\boxed{\varepsilon \rhd \varepsilon}$

$$
\varepsilon_1 \rhd \varepsilon_2 \;:=\; \varepsilon_1 \cup \varepsilon_2 \qquad \text{Sequential Composition}
$$

Fig. 3. Operators on qualifiers and effects. We often leave the context implicit (marked as gray).

stripped-down effect system corresponding to a trivial effect quantale just tracking whether an effect is induced on reachable variables, effectively making effects just another qualifier (i.e., a set of variables and locations) in the typing judgment. This effect system is sufficient for calculating granular effect dependencies. This version also lacks a ⊥ qualifier for untracked values, and recursive λ-abstractions.[3] To keep the discussion focused and on point, we omitted those features which do not add much to the discussion of the core ideas apart from additional proof cases.

2.2 Examples

Here we use a few examples from Bao et al. [2021] to motivate the direct-style λ^*_ε-calculus as the base system. The basic idea is to track which other values are reachable from a given expression's result. For example, the following expression allocates a reference cell of integers and binds it to x:

```
val x = new Ref(42)        // : Ref[Int]^{x}  {w}
```

Therefore, by reflexivity, the type of x has a qualifier (*i.e.* a set of variables) that contains itself. The allocation also induces an effect over an allocator w for allocating memory resources. We can further bind x to a new variable y, which induces no effect (indicated by the empty set):

```
val y = x                  // : Ref[Int]^{y}   ∅ ← lazy assignment (this work)
                           // : Ref[Int]^{x,y} ∅ ← eager assignment (Bao et al.)
```

As in Wei et al. [2023a], the system considered here is "lazy" with respect to qualifier assignment, in the sense that it assigns minimal qualifiers. For example, y reaches itself, just like x does, but in this particular context, we can also deduce that y indirectly reaches x, since it is an alias. This provides a certain degree of lightweight, i.e., quantifier-free, reachability polymorphism [Wei et al. 2023a]. We use the notation $\{y\}^* = \{x,y\}$ for the transitively closed reachability set in a given context (Figure 3). In contrast, Bao et al. use an "eager" qualifier assignment, *e.g.*, they would always assign the transitively closed qualifier.

We refer readers to Section 2 of Bao et al. [2021] for more illustrative examples of reachability types. Section 2 of the main paper [Bračevac et al. 2023] also provides examples after adapting reachability types to the graph IR.

[3] Modulo the (straightforward) addition of effects, the system presented here closely follows the mechanized "overlap lazy" variant found at https://github.com/tiarkrompf/reachability/tree/main/base/lambda_star_overlap_lazy.

2.3 Syntax

Figure 2 shows the syntax of $\lambda^{*}_{\varepsilon}$ which is based on the simply-typed λ-calculus with mutable references and subtyping. We denote general term variables by the meta variables x, y, z, and reserve ℓ, w for store locations.

Terms consist of constants of base types, variables, functions $\lambda x.t$, function applications, reference allocations, dereferences, assignments, and **let**-expressions.

Reachability qualifiers p, q, r are finite sets of variables and store locations. For readability, we often drop the set notation for qualifiers and write them down as comma-separated lists of atoms.

We distinguish ordinary types T from qualified types T^{q}, where the latter annotates a qualifier q to an ordinary type T. The types consist of base types B (*e.g.*, Int, Unit), dependent function types $(x : T^{q}) \to^{\varepsilon} S^{p}$, where both argument and return type are qualified. The codomain S^{p} may depend on the argument x in its qualifier and type. Function types carry an annotation ε for its latent effect, which is a set of variables and locations, akin to qualifiers.

For simplicity, mutable reference types Ref B can only store values of base types. We could also permit forms of nested references, by adding a flow-sensitive effect system [Bao et al. 2021].

An *observation* φ is a finite set of variables which is part of the term typing judgment (Section 2.4). It specifies which variables and locations in the typing context Γ and store typing Σ are observable. The former assigns qualified typing assumptions to variables.

2.4 Statics

The term typing judgment $[\Sigma \mid \Gamma]^{\varphi} \vdash t : T^{q}\ \varepsilon$ in Figure 2 states that term t has qualified type T^{q} and may induce effect ε, and may only access the typing assumptions of Γ observable by φ. One may think of t as a computation that incurs effect ε and yields a result value of type T aliasing no more than q, if it terminates.

Different from Bao et al. [2021], we internalize the filter φ as part the typing relation. Alternatively, we could formulate the typing judgment without internalizing φ, and instead have an explicit context filter operation $\Gamma^{\varphi} := \{x : T^{q} \in \Gamma \mid q, x \subseteq \varphi\}$ for restricting the context in subterms, just like Bao et al. [2021] which loosely takes inspiration from substructural type systems. Internalizing φ (1) makes observability an explicit notion, which facilitates reasoning about separation and overlap, and (2) greatly simplifies the Coq mechanization. Context filtering is only needed for term typing, but not for subtyping, so as to keep the formalization simple.

2.4.1 *Functions and Lightweight Polymorphism.*

Function typing (t-abs) implements the observable separation guarantee, *i.e.*, the body t can only observe what the function type's qualifier q specifies, plus the argument x, and is otherwise oblivious to anything else in the environment. We model this by setting the observation to q, x, f when typing the body. Thus, its observation q at least includes the free variables of t. To ensure well-scopedness, q must be a subset of the observation φ on the outside. In essence, a function type *implicitly* quantifies over anything that is not observed by q, achieving a lightweight form of qualifier polymorphism.

2.4.2 *Dependent Application, Separation and Overlap.*

Function applications (t-app) are qualifier-dependent in that the result qualifier can depend on the argument.

Function applications also establish an *observable separation* between the argument reachable set p and the function reachable set q, as denoted as $p{*} \cap q{*}$. The intersection between $p{*}$ and $q{*}$ specifies the permitted overlap. We are careful to intersect the transitive reachability closure (a.k.a. saturated version, Figure 3) of the two qualifiers. This is necessary in the lazy reachability assignment, because we might miss common, indirect overlap between the sets otherwise. If the intersection declared in the function type is empty, then it means complete separation between the argument and the entities observed by the function from the environment.

2.4.3 Qualifier Substitution. The base substitution operation $q[p/x]$ of qualifiers for variables is defined in Figure 3, and we use it along with its homomorphic extension to types in dependent function application. Substitution replaces the variable with the given qualifier, if present in the target.

2.4.4 Effects. Our effect system is a simple flow-insensitive instantiation of Gordon [2021]'s effect quantale system. An effect ε denotes the set of variables/locations that might be used during the computation. For a compound term, the final effect is computed by composing the effects of sub-terms with the intrinsic effect of this term. For example, the effect of assignments has two parts: (1) ε_1, ε_2 the effects of sub-terms, and (2) q the variables/locations being modified. The final effect is obtained by composing these effects.

Although the typing rules presented in Figure 2 pretend to use the sequential effect composition operator $\rhd$, its definition $\cup$ computes an upper bound of two effects and is *not* flow-sensitive (Figure 3), *i.e.* the composed effect is not sensitive to the order of composition. This simple instantiation is sufficient for deriving dependencies (cf. Section 5).

2.4.5 Mutable References. Slightly different from Bao et al. [2021], the allocation $\mathbf{ref}_{t_1}\ t_2$ (T-REF) additionally takes a term t_1 that has an Alloc primitive type. An allocation induces an effect over aliases of t_1, which is recorded as the composed term effect.

The typing rule of reference allocation (T-REF), read (T-!), and write (T-:=) work with reference types whose inner referent values are base values. This is sufficient for understanding the core ideas of the graph IR. It is nevertheless possible to extend the base system with nested references (*e.g.* allowing storing functions) by a flow-sensitive effect system, as shown by Bao et al. [2021]. The subtyping of references is invariant in the referent type.

2.4.6 Subtyping. We distinguish subtyping between qualifiers q, ordinary types T, and qualified types T^{q}, where the latter two are mutually dependent. Subtyping is assumed to be well-scoped under the typing context Γ and store Σ, *i.e.*, types and qualifiers mention only variables/locations bound in Γ and Σ, and so do its typing assumptions. Qualified subtyping (SQE-SUB) just forwards to the other two judgments for scaling the type, qualifier, and effect respectively.

Qualifier Subtyping. Qualifier subtyping includes the subset relation (Q-SUB), which resort to the subset relation since qualifiers are sets. Since effects are just qualifiers, we use the same subtyping relation for subeffecting.

Ordinary Subtyping. Subtyping rules for base types (S-BASE), reference types (S-REF), and function types (S-FUN) are standard modulo qualifiers. Reflexivity and transitivity are both admissible for subtyping on ordinary and qualified types. Function types are contravariant in the domain, and covariant in the codomain and effect, as usual. Due to dependency in the codomain, we are careful to extend the context with the smaller argument type.

2.5 Dynamics

The single-step, call-by-value (CBV) for λ^*_ε (Figure 4) is standard. To bridge the gap to monadic normal forms later on, we model stores as sequences of mutable let bindings. One may view computations as syntactic sequences of let bindings (think already evaluated graph nodes) followed by a redex. Another difference to standard treatments is that reference allocation takes the allocation-capability variable ω as an explicit extra argument. This design makes the treatment of effectful operations uniform, in the sense that an operation always induces an effect on some operand.

Stores, Evaluation Contexts

$$\begin{array}{rcl} \sigma & ::= & \varnothing \mid \sigma,\ \textbf{let}_{\mathrm{s}}\ \ell = \textbf{ref}_{\omega}\ v \\ E & ::= & \square \mid E\ t \mid v\ E \mid \textbf{ref}_{E}\ t \mid \textbf{ref}_{v}\ E \mid\ !E \mid E := t \mid v := E \mid \textbf{let}\ x = E\ \textbf{in}\ t \end{array}$$

Well-Formed Stores $\boxed{[\Sigma \mid \Gamma]^{\varphi} \vdash \sigma}$

$$\frac{}{[\varnothing \mid \Gamma]^{\varphi} \vdash \varnothing} \qquad \frac{[\Sigma \mid \Gamma]^{\varphi} \vdash \sigma \qquad [\Sigma \mid \Gamma]^{\varphi} \vdash v : B^{\varnothing}\ \varnothing \qquad \ell \notin \mathrm{dom}(\Sigma)}{[\Sigma, \ell : \mathrm{Ref}\ B^{\varnothing} \mid \Gamma]^{\varphi} \vdash \sigma,\ \textbf{let}_{\mathrm{s}}\ \ell = \textbf{ref}\ v}$$

Reduction Rules $\boxed{\sigma \mid t \longrightarrow_{\mathrm{v}} \sigma \mid t}$

$$\begin{array}{rcll} \sigma \mid E[\,(\lambda x.t)\ v\,] & \longrightarrow_{\mathrm{v}} & \sigma \mid E[\,t[v/x]\,] & (\beta) \\ \sigma \mid E[\,\textbf{let}\ x = v\ \textbf{in}\ t\,] & \longrightarrow_{\mathrm{v}} & \sigma \mid E[\,t[v/x]\,] & (\textsc{let}) \\ \sigma \mid E[\,\textbf{ref}_{\omega}\ v\,] & \longrightarrow_{\mathrm{v}} & \sigma,\ \textbf{let}_{\mathrm{s}}\ \ell = \textbf{ref}_{\omega}\ v \mid E[\,\ell\,] & (\textsc{ref}) \\ & & \ell \notin \mathrm{dom}(\sigma) & \\ \sigma,\ \textbf{let}_{\mathrm{s}}\ \ell = \textbf{ref}_{\omega}\ v,\ \sigma' \mid E[\,!\ell\,] & \longrightarrow_{\mathrm{v}} & \sigma,\ \textbf{let}_{\mathrm{s}}\ \ell = \textbf{ref}_{\omega}\ v,\ \sigma' \mid E[\,v\,] & (\textsc{deref}) \\ \sigma,\ \textbf{let}_{\mathrm{s}}\ \ell = \textbf{ref}_{\omega}\ v,\ \sigma' \mid E[\,\ell := v'\,] & \longrightarrow_{\mathrm{v}} & \sigma,\ \textbf{let}_{\mathrm{s}}\ \ell = \textbf{ref}_{\omega}\ v',\ \sigma' \mid E[\,\mathsf{unit}\,] & (\textsc{assign}) \end{array}$$

Fig. 4. Standard call-by-value reduction for $\lambda^{*}_{\varepsilon}$.

The allocation capability ω is a base constant for allocation of base type Alloc, we consider an initial store with $\textbf{let}_{\mathrm{s}}\ w = \omega$, and $w : \mathsf{Alloc}^{\varnothing}$.

2.6 Metatheory

The $\lambda^{*}_{\varepsilon}$-calculus exhibits syntactic type soundness which we prove by standard progress and preservation properties (Theorems 2.11 and 2.12). Type soundness implies the preservation of separation corollary (Corollary 2.13) as set forth by Bao et al. [2021] for their λ^{*}-calculus. It is a memory property certifying that the results of well-typed $\lambda^{*}_{\varepsilon}$ terms with disjoint qualifiers indeed never alias.

The metatheory of the $\lambda^{*}_{\varepsilon}$-calculus closely follows the "lazy overlap" variant of λ^{*}-calculus, which has been mechanized.[4] The major difference lies in the addition of a simple effect system, which does not change the metatheory significantly, other than carrying an extra qualifier for effects in judgments. Below, we discuss key lemmas required for the type soundness proof.

2.6.1 Observability Properties. Reasoning about substitutions and their interaction with overlap/separation in preservation lemmas requires that the qualifiers assigned by term typing are observable. The following lemmas are proved by induction over the respective typing derivations:

Lemma 2.1 (Observability Invariant). *Term typing always assigns observable qualifiers and effects,* i.e., *if* $[\Sigma \mid \Gamma]^{\varphi} \vdash t : T^{q}\ \varepsilon$*, then* $q, \varepsilon \subseteq \varphi$.

Well-typed values cannot observe anything about the context beyond their assigned qualifier:

Lemma 2.2 (Tight Observability for Values). *If* $[\Sigma \mid \Gamma]^{\varphi} \vdash v : T^{q}\ \varepsilon$*, then* $[\Sigma \mid \Gamma]^{q} \vdash v : T^{q}\ \varnothing$.

It is easy to see that any observation for a function $\lambda x.t$ will at least track the free variables of the body t.

[4] The mechanization can be found at https://github.com/tiarkrompf/reachability/tree/main/base/lambda_star_overlap_lazy.

2.6.2 Weakening and Narrowing Lemmas. The λ^*_ε calculus has standard weakening and narrowing lemmas.

Lemma 2.3 (Subtyping Weakening).

$$\frac{\Sigma \mid \Gamma \vdash p <: q \qquad \Gamma' \supseteq \Gamma \qquad \Sigma' \supseteq \Sigma}{\Sigma' \mid \Gamma' \vdash p <: q}$$

$$\frac{\Sigma \mid \Gamma \vdash S <: T \qquad \Gamma' \supseteq \Gamma \qquad \Sigma' \supseteq \Sigma}{\Sigma' \mid \Gamma' \vdash S <: T}$$

$$\frac{\Sigma \mid \Gamma \vdash S^p\ \varepsilon_1 <: T^q\ \varepsilon_2 \qquad \Gamma' \supseteq \Gamma \qquad \Sigma' \supseteq \Sigma}{\Sigma' \mid \Gamma' \vdash S^p\ \varepsilon_1 <: T^q\ \varepsilon_2}$$

Proof. Weakening on qualifier subtyping trivially follows from its definition. The others are proved by mutual induction over the respective derivations. □

Lemma 2.4 (Weakening).

$$\frac{[\Sigma \mid \Gamma]^{\varphi} \vdash t : T^q\ \varepsilon \qquad \Gamma' \supseteq \Gamma \qquad \Sigma' \supseteq \Sigma \qquad \varphi' \supseteq \varphi}{[\Sigma' \mid \Gamma']^{\varphi'} \vdash t : T^q\ \varepsilon}$$

Proof. By induction over the term typing derivation, using Lemma 2.3 where appropriate. □

Lemma 2.5 (Subtyping Narrowing).

$$\frac{\Sigma \mid \Gamma, x : V^p, \Gamma' \vdash q <: r \qquad \Sigma \mid \Gamma \vdash U^o\ \varepsilon_1 <: V^p\ \varepsilon_2}{\Sigma \mid \Gamma, x : U^o, \Gamma' \vdash q <: r}$$

$$\frac{\Sigma \mid \Gamma, x : V^p, \Gamma' \vdash S <: T \qquad \Sigma \mid \Gamma \vdash U^o\ \varepsilon_1 <: V^p\ \varepsilon_2}{\Sigma \mid \Gamma, x : U^o, \Gamma' \vdash S <: T}$$

$$\frac{\Sigma \mid \Gamma, x : V^p, \Gamma' \vdash S^q\ \varepsilon_3 <: T^r\ \varepsilon_4 \qquad \Sigma \mid \Gamma \vdash U^o\ \varepsilon_1 <: V^p\ \varepsilon_2}{\Sigma \mid \Gamma, x : U^o, \Gamma' \vdash S^q\ \varepsilon_3 <: T^r\ \varepsilon_4}$$

Proof. By mutual induction over the respective derivations. □

Lemma 2.6 (Narrowing).

$$\frac{[\Sigma \mid \Gamma, x : V^p, \Gamma']^{\varphi} \vdash t : T^q\ \varepsilon_1 \qquad \Sigma \mid \Gamma \vdash U^o\ \varepsilon_2 <: V^p\ \varepsilon_3}{[\Sigma \mid \Gamma, x : U^o, \Gamma']^{\varphi} \vdash t : T^q\ \varepsilon_1}$$

Proof. By induction over the term-typing derivation, using Lemma 2.5 where appropriate. □

2.6.3 Substitution Lemmas. We consider type soundness for closed terms and apply "top-level" substitutions, *i.e.*, substituting closed values with qualifiers that do not contain term variables, but only store locations. The proof of the substitution lemma critically relies on the distributivity of substitution and the qualifier intersection operator for checking overlap, which is required to proceed in the (t-app) case:

Lemma 2.7 (Top-Level Substitutions Distribute with Overlap).

$$\frac{x : T^{q} \in \Gamma \qquad \theta = [p/x] \qquad p, q \subseteq \mathit{dom}(\Sigma) \qquad p \cap \varphi \subseteq q \qquad r, r' \subseteq \varphi \qquad r = r* \qquad r' = r'*}{(r \cap r')\theta = r\theta \cap r'\theta}$$

Qualifier substitution does not generally distribute with set intersection, due to the problematic case when the substituted variable x occurs in only one of the saturated sets r and r'. Distributivity holds if (1) we ensure that what is observed about the qualifier p we substitute for x is bounded by what the context observes about x, *i.e.*, $p \cap \varphi \subseteq q$ for $x : T^{q} \in \Gamma$, and (2) p, q are top-level as above. Furthermore, we require that the intersected qualifiers r and r' are reachability saturated, which is given in the context of (t-app).

Lemma 2.8 (Top-Level Substitution for Qualifier/Effect Subtyping).

$$\frac{\Sigma \mid \Gamma, x : T^{q} \vdash p <: r \qquad q, q' \subseteq \mathrm{dom}(\Sigma) \qquad \theta = [q'/x] \qquad \Sigma \mid \Gamma, x : T^{q}\ \mathrm{ok}}{\Sigma \mid \Gamma\theta \vdash p\theta <: r\theta}$$

Proof. By the fact that substitution is monotonic w.r.t. subset inclusion $\subseteq$ and qualifier/effect subtyping being that relation by definition. □

Lemma 2.9 (Top-Level Substitution for Subtyping).

$$\frac{\Sigma \mid \Gamma, x : S^{q} \vdash T <: U \qquad q, p \subseteq \mathrm{dom}(\Sigma) \qquad \theta = [p/x]}{\Sigma \mid \Gamma\theta \vdash T\theta <: U\theta}$$

$$\frac{\Sigma \mid \Gamma, x : S^{q} \vdash T^{p}\ \varepsilon_1 <: U^{q}\ \varepsilon_2 \qquad q, p \subseteq \mathrm{dom}(\Sigma) \qquad \theta = [p/x]}{\Sigma \mid \Gamma\theta \vdash T^{p}\theta\ \varepsilon_1\theta <: U^{q}\theta\ \varepsilon_2\theta}$$

Proof. By mutual induction over the respective subtyping derivations, using Lemma 2.8 where appropriate. □

In the type preservation proof, β-reduction substitutes a function parameter for some value, which requires a carefully formulated substitution lemma:

Lemma 2.10 (Top-Level Term Substitution).

$$\frac{[\Sigma \mid \Gamma, x : S^{p \cap r}]^{\varphi} \vdash t : T^{q}\ \varepsilon \qquad [\Sigma \mid \varnothing]^{p} \vdash v : S^{p}\ \varnothing \qquad \theta = [p/x] \qquad p \subseteq \mathit{dom}(\Sigma) \qquad p \cap \varphi \subseteq p \cap r}{[\Sigma \mid \Gamma\theta]^{\varphi\theta} \vdash t[v/x] : (T^{q}\ \varepsilon)\theta}$$

Proof. By induction over the derivation $[\Sigma \mid \Gamma, x : S^{p \cap r}]^{\varphi} \vdash t : T^{q}\ \varepsilon$. Most cases are straightforward, exploiting that qualifier substitution is monotonous w.r.t. $\subseteq$ and that the substitute p for x consists of store locations only. The case (t-app) critically requires Lemma 2.7 for $(p \cap q)\theta = p\theta \cap q\theta$ in the induction hypothesis. The case (t-sub) requires the substitution lemma for subtyping (Lemma 2.9). □

Just as in Lemma 2.7 above, the substitution lemma imposes the observability condition $p \cap \varphi \subseteq p \cap r$, *i.e.*, t observes nothing more about v's reachability set than its assumption about x, and it is oblivious of $p \setminus r$. That is to say, substitution "grows" the parameter in (T-APP) with overlap between p and the function qualifier r, growing the result by $p \setminus r$, realizing implicit polymorphism over qualifiers.

2.6.4 Main Soundness Result.

Theorem 2.11 (Progress). *If* $[\Sigma \mid \varnothing]^{\mathrm{dom}(\Sigma)} \vdash t : T^{q}\ \varepsilon$*, then either t is a value, or for any store σ where* $[\Sigma \mid \varnothing]^{\mathrm{dom}(\Sigma)} \vdash \sigma$*, there exists a term t' and store σ' such that* $\sigma \mid t \longrightarrow_{\mathrm{v}} \sigma' \mid t'$.

Proof. By induction over the derivation $[\Sigma \mid \varnothing]^{\mathrm{dom}(\Sigma)} \vdash t : T^{q}\ \varepsilon$. □

Similar to [Bao et al. 2021], reduction preserves types up to qualifier growth by fresh allocations:

Theorem 2.12 (Preservation).

$$\frac{[\Sigma \mid \varnothing]^{\mathrm{dom}(\Sigma)} \vdash t : T^{q}\ \varepsilon \qquad [\Sigma \mid \varnothing]^{\mathrm{dom}(\Sigma)} \vdash \sigma \qquad \sigma \mid t \longrightarrow_{\mathrm{v}} \sigma' \mid t'}{\exists \Sigma' \supseteq \Sigma.\ \exists p \subseteq \mathrm{dom}(\Sigma' \setminus \Sigma).\quad [\Sigma' \mid \varnothing]^{\mathrm{dom}(\Sigma')} \vdash t' : T^{q,p}\ \varepsilon, p \qquad [\Sigma' \mid \varnothing]^{\mathrm{dom}(\Sigma')} \vdash \sigma'}$$

Proof. By induction over the derivation $[\Sigma \mid \varnothing]^{\mathrm{dom}(\Sigma)} \vdash t : T^{q}\ \varepsilon$. Most of the cases are straightforward. We discuss the beta reduction case of (T-APP) where the substitution lemma (Lemma 2.10) needs to be applied. To make the proof simpler, we assume explicit congruence reduction rules here.

In this case, we have $t = (\lambda x.t_0)\ v$ and their typings by induction hypotheses: $[\Sigma \mid \varnothing]^{\mathrm{dom}(\Sigma)} \vdash \lambda x.t_0 : \left(x : T^{p* \cap q*} \to^{\varepsilon_3} U^{r}\right)^{q}\ \varepsilon_1$ and $[\Sigma \mid \varnothing]^{\mathrm{dom}(\Sigma)} \vdash v : T^{p}\ \varepsilon_2$. We need to show

$$[\Sigma \mid \varnothing]^{\mathrm{dom}(\Sigma)} \vdash t_0[v/x] : (U^{r}\ \varepsilon, p)[p/x].$$

Inverting the tying of the lambda value, we have the body term t_0 typing

$$[\Sigma \mid x : T'^{p'}]^{q' \cup \{x\}} \vdash t_0 : U'^{r'}\ \varepsilon',$$

and

$$T^{q \cap p} <: T'^{p'}, q' <: q, \text{ and } \Sigma \mid x : T^{p* \cap q*} \vdash U'^{r'} \varepsilon' <: U^{r} \varepsilon, p.$$

By narrowing the context and weakening the filter, we obtain a body term typing that is amenable to apply the substitution lemma (Lemma 2.10):

$$[\Sigma \mid x : T^{p* \cap q*}]^{q \cup \{x\}} \vdash t_0 : U'^{r'}\ \varepsilon'.$$

Then after applying Lemma 2.10, we use (T-SUB) to up-cast the result type and effect, which proves the goal.

□

Corollary 2.13 (Preservation of Separation). *Interleaved executions preserve types and disjointness:*

$$\frac{\begin{array}{lll} [\Sigma \mid \varnothing]^{\mathrm{dom}(\Sigma)} \vdash t_1 : T_1^{q_1}\ \varepsilon_1 & \sigma\ \mid t_1 \longrightarrow_{\mathrm{v}} t_1' \mid \sigma' & [\Sigma \mid \varnothing]^{\mathrm{dom}(\Sigma)} \vdash \sigma \\ [\Sigma \mid \varnothing]^{\mathrm{dom}(\Sigma)} \vdash t_2 : T_2^{q_2}\ \varepsilon_2 & \sigma' \mid t_2 \longrightarrow_{\mathrm{v}} t_2' \mid \sigma'' & q_1 \cap q_2 \subseteq \varnothing \end{array}}{\begin{array}{lll} \exists p_1\ p_2\ \varepsilon_1'\ \varepsilon_2'\ \Sigma'\ \Sigma''. & [\Sigma'\ \mid \varnothing]^{\mathrm{dom}(\Sigma')} \vdash t_1' : T_1^{p_1}\ \varepsilon_1' & \Sigma'' \supseteq \Sigma' \supseteq \Sigma \\ & [\Sigma'' \mid \varnothing]^{\mathrm{dom}(\Sigma'')} \vdash t_2' : T_2^{p_2}\ \varepsilon_2' & p_1 \cap p_2 \subseteq \varnothing \end{array}}$$

Proof. By sequential application of preservation (Theorem 2.12) and the fact that a reduction step increases the assigned qualifier by at most a fresh new location, thus preserving disjointness. □

Introductions, Store Terms, Evaluation Contexts

$$\begin{array}{lcl} \iota & ::= & \lambda x.t \mid c \mid \mathbf{ref}_{\ell}\ \ell \\ \sigma & ::= & \varnothing \mid \sigma,\ \mathbf{let}_{\mathsf{s}}\ \ell = \iota \\ E & ::= & \square \mid E\ t \mid \ell\ E \mid \mathbf{ref}_{E}\ t \mid \mathbf{ref}_{\ell}\ E \mid\ !E \mid E := t \mid \ell := E \mid \mathbf{let}\ x = E\ \mathbf{in}\ t \end{array}$$

Well-Formed Store Entries and Stores $\boxed{[\Sigma \mid \Gamma]^{\varphi} \vdash \ell : \iota \in \sigma}$ $\boxed{[\Sigma \mid \Gamma]^{\varphi} \vdash \sigma}$

$$\frac{\Sigma(\ell) = \mathsf{Ref}\ B^{\varnothing} \qquad [\Sigma \mid \Gamma]^{\varphi} \vdash \ell' : B^{\varnothing}\ \varnothing \qquad \Sigma(w) = \mathsf{Alloc}^{\varnothing} \qquad \sigma(w) = \omega}{[\Sigma \mid \Gamma]^{\varphi} \vdash \ell : \mathbf{ref}_{w}\ \ell' \in \sigma}$$

$$\frac{\Sigma(\ell) = T^{q} \qquad [\Sigma \mid \Gamma]^{\varphi} \vdash \iota : T^{q}\ \varnothing \qquad \forall \ell, w.\ \iota \neq \mathbf{ref}_{w}\ \ell}{[\Sigma \mid \Gamma]^{\varphi} \vdash \ell : \iota \in \sigma}$$

$$\frac{|\Sigma| = |\sigma| \qquad \left([\Sigma \mid \Gamma]^{\varphi} \vdash \ell : \iota \in \sigma\right)_{\mathbf{let}_{\mathsf{s}}\ \ell = \iota \in \sigma}}{[\Sigma \mid \Gamma]^{\varphi} \vdash \sigma}$$

Reduction Rules $\boxed{\sigma \mid t \longrightarrow_{\mathbf{sv}} \sigma \mid t}$

$$\begin{array}{rcll} \sigma,\ \mathbf{let}_{\mathsf{s}}\ \ell_1 = \lambda x.t,\ \sigma' \mid E[\ \ell_1\ \ell_2\ ] & \longrightarrow_{\mathbf{sv}} & \sigma,\ \mathbf{let}_{\mathsf{s}}\ \ell_1 = \lambda x.t,\ \sigma' \mid E[\ t[\ell_2/x]\ ] & (\beta) \\ \sigma \mid E[\ \mathbf{let}\ x = \ell\ \mathbf{in}\ t\ ] & \longrightarrow_{\mathbf{sv}} & \sigma \mid E[\ t[\ell/x]\ ] & (\textsc{let}) \\ \sigma \mid E[\ \iota\ ] & \longrightarrow_{\mathbf{sv}} & \sigma, \mathbf{let}_{\mathsf{s}}\ \ell = \iota \mid E[\ \ell\ ] & (\textsc{intro}) \\ & & \ell \notin \mathrm{dom}(\sigma) & \\ \sigma,\ \mathbf{let}_{\mathsf{s}}\ \ell = \mathbf{ref}_{w}\ \ell',\ \sigma' \mid E[\ !\ell\ ] & \longrightarrow_{\mathbf{sv}} & \sigma,\ \mathbf{let}_{\mathsf{s}}\ \ell = \mathbf{ref}_{w}\ \ell',\ \sigma' \mid E[\ \ell'\ ] & (\textsc{deref}) \\ \sigma,\ \mathbf{let}_{\mathsf{s}}\ \ell = \mathbf{ref}_{w}\ \ell',\ \sigma' \mid E[\ \ell := \ell''\ ] & \longrightarrow_{\mathbf{sv}} & \sigma,\ \mathbf{let}_{\mathsf{s}}\ \ell = \mathbf{ref}_{w}\ \ell'',\ \sigma' \mid E[\ \mathsf{unit}\ ] & (\textsc{assign}) \end{array}$$

Fig. 5. Call-by-value reduction for $\lambda^{*}_{\varepsilon}$ with store-allocated values.

3 THE DIRECT-STYLE $\lambda^{*}_{\varepsilon}$-CALCULUS WITH STORE-ALLOCATED VALUES

As a first step towards transitioning into monadic normal form, we refine the previous system's operational semantics into one that has all values in the store, *i.e.*, substitution becomes variable renaming, because all intermediate results are named and bound in the store. We keep the same type system as before and show its soundness with respect to the refined operational semantics with store-allocated values.

3.1 Syntax

We introduce a slight change to the syntax of $\lambda^{*}_{\varepsilon}$ (Figure 2) that does not affect the typing rules, namely changing what constitutes a value and re-categorizing former values and reference allocations as "introductions" ι for store-bound entities:

$$\begin{array}{rlr} v & ::= \ell & \text{Values} \\ \iota & ::= c \mid \lambda x.t \mid \mathbf{ref}_{\ell}\ \ell & \text{Introductions} \\ t & ::= \cdots \mid v \mid \iota & \text{Terms} \\ \sigma & ::= \varnothing \mid \sigma,\ \mathbf{let}_{\mathsf{s}}\ \ell = \iota & \text{Stores} \end{array}$$

Both mutable references and immutable constants are part of the store now, and we can discern by types and context relations whether a location ℓ may be mutated at runtime or not.

Since all constants are store-bound, we also expect that the first operand of **ref** is a location binding the allocation capability/constant ω.

3.2 Dynamics

Figure 5 shows the operational semantics for λ^*_ε with store-allocated values. All elimination forms, now operate on store-bound introductions. For instance, the function application rule (β) replaces the call with the body of the function stored at ℓ_1, and passes a location ℓ_2 pointing to the argument of the call. Substitution on terms simply becomes a renaming of a variable to a store location. The new rule (INTRO) replaces the previous rule (REF), generalizing it to commit any introduction into the store at a fresh location.

3.3 Metatheory

Since the type system has not changed, we can reuse most of the results developed in Section 2.6.

LEMMA 3.1 (TOP-LEVEL TERM SUBSTITUTION).

$$\frac{[\Sigma \mid \Gamma, x : S^{p\cap r}]^{\varphi} \vdash t : T^{q}\ \varepsilon \qquad [\Sigma \mid \varnothing]^{p} \vdash \ell : S^{p}\ \varnothing \qquad \theta = [p/x] \qquad p \subseteq dom(\Sigma) \qquad p \cap \varphi \subseteq p \cap r}{[\Sigma \mid \Gamma\theta]^{\varphi\theta} \vdash t[\ell/x] : (T^{q}\ \varepsilon)\theta}$$

PROOF. This is a special case of Lemma 2.10. □

THEOREM 3.2 (PRESERVATION).

$$\frac{[\Sigma \mid \varnothing]^{\mathrm{dom}(\Sigma)} \vdash t : T^{q}\ \varepsilon \qquad [\Sigma \mid \varnothing]^{\mathrm{dom}(\Sigma)} \vdash \sigma \qquad \sigma \mid t \longrightarrow_{\mathrm{sv}} \sigma' \mid t'}{\exists \Sigma' \supseteq \Sigma.\ \exists p \subseteq \mathrm{dom}(\Sigma' \setminus \Sigma). \quad [\Sigma' \mid \varnothing]^{\mathrm{dom}(\Sigma')} \vdash t' : T^{q,p}\ \varepsilon, p \qquad [\Sigma' \mid \varnothing]^{\mathrm{dom}(\Sigma')} \vdash \sigma'}$$

PROOF. By induction over the derivation $[\Sigma \mid \varnothing]^{\mathrm{dom}(\Sigma)} \vdash t : T^{q}\ \varepsilon$. The proof is similar to the proof for Theorem 2.12, with the difference that typing evidence for operands needs to be extracted from the well-formed store σ. □

Finally, the preservation of separation Corollary 2.13 continues to hold in this system, with exactly the same proof.

4 MONADIC NORMAL FORM

The penultimate step towards deriving the graph IR is restricting the λ^*_ε language to *monadic normal form* (MNF), called λ^*_{M} (Figure 6). We establish soundness of λ^*_{M} by (1) showing that λ^*_ε's reduction relation with store-allocated values ($\longrightarrow_{\mathrm{sv}}$, Figure 5) preserves MNF, (2) specifying provably type-preserving translations between both languages, so that (3) we can resort to the previous section's soundness result for λ^*_ε.

4.1 Syntax

We make use of the syntactic category of names in places where both variables and locations are permitted, written in typewriter font.

MNF (Figure 6) is characterized by having all intermediate results and subterms of expressions let-bound to variable names. Unlike A-normal form (ANF), which has strictly flat sequences of let bindings with primitive operations, MNF permits binding nested computations. A (directed, acyclic) graph can be read from graph terms g, by regarding let bindings as introducing a name for either (1) a primitive graph node labelled with a primitive operation drawn from n, or (2) naming a nested subgraph g. Variable occurrences in bound nodes correspond to edges pointing to the let binding in scope.

Monadic Normal Form $\boxed{\lambda^*_{\mathsf{M}}}$

$$\begin{array}{llll}
\mathsf{x},\mathsf{y},\mathsf{z} & ::= & x \mid \ell & \text{Names} \\
\iota & ::= & c \mid \lambda x.g \mid \mathbf{ref}_\ell\ \ell & \text{Introductions} \\
v & ::= & \ell & \text{Values} \\
n & ::= & \iota \mid \mathsf{x}\ \mathsf{x} \mid \mathbf{ref}_{\mathsf{x}}\ \mathsf{x} \mid !\ \mathsf{x} \mid \mathsf{x} := \mathsf{x} & \text{Graph Nodes} \\
g & ::= & \mathsf{x} \mid \mathbf{let}\ x = b\ \mathbf{in}\ g & \text{Graph Terms} \\
b & ::= & n \mid g & \text{Bindings}
\end{array}$$

MNF Typing $\boxed{[\Sigma \mid \Gamma]^{\varphi} \vdash_{\mathsf{M}} n : T^{q}\ \varepsilon}$ $\boxed{[\Sigma \mid \Gamma]^{\varphi} \vdash_{\mathsf{M}} g : T^{q}\ \varepsilon}$ $\boxed{[\Sigma \mid \Gamma]^{\varphi} \vdash_{\mathsf{M}} b : T^{q}\ \varepsilon}$

$$\frac{c \in B}{[\Sigma \mid \Gamma]^{\varphi} \vdash_{\mathsf{M}} c : B^{\varnothing}\ \varnothing}\ (\text{N-CST}) \qquad \frac{\begin{array}{c}\mathsf{x} : (\mathsf{Ref}\ B)^{q} \in [\Sigma \mid \Gamma]^{\varphi} \\ \mathsf{y} : B^{\varnothing} \in [\Sigma \mid \Gamma]^{\varphi}\end{array}}{[\Sigma \mid \Gamma]^{\varphi} \vdash_{\mathsf{M}} \mathsf{x} := \mathsf{y} : \mathsf{Unit}^{\varnothing}\ \mathsf{x}}\ (\text{N-:=})$$

$$\frac{\begin{array}{c}[\Sigma \mid \Gamma, x : T^{p}]^{q,x} \vdash_{\mathsf{M}} g : U^{r}\ \varepsilon \\ q \subseteq \varphi\end{array}}{[\Sigma \mid \Gamma]^{\varphi} \vdash_{\mathsf{M}} \lambda x.g : (x : T^{p} \to^{\varepsilon} U^{r})^{q}\ \varnothing}\ (\text{N-ABS}) \qquad \frac{\mathsf{x} : T^{q} \in [\Sigma \mid \Gamma]^{\varphi}}{[\Sigma \mid \Gamma]^{\varphi} \vdash_{\mathsf{M}} \mathsf{x} : T^{\mathsf{x}}\ \varnothing}\ (\text{G-RET})$$

$$\frac{\begin{array}{c}\mathsf{x} : \left(z : T^{p*\cap q*} \to^{\varepsilon} U^{r}\right)^{q} \in [\Sigma \mid \Gamma]^{\varphi} \\ \mathsf{y} : T^{p} \in [\Sigma \mid \Gamma]^{\varphi} \quad \theta = [p/z] \\ z \notin \mathrm{fv}(U) \quad \varepsilon \subseteq q, z \quad r \subseteq \varphi, z\end{array}}{[\Sigma \mid \Gamma]^{\varphi} \vdash_{\mathsf{M}} \mathsf{x}\ \mathsf{y} : (U^{r}\ \varepsilon)\theta}\ (\text{N-APP}) \qquad \frac{\begin{array}{c}[\Sigma \mid \Gamma]^{\varphi} \vdash_{\mathsf{M}} b : S^{p}\ \varepsilon_1 \\ [\Sigma \mid \Gamma, x : S^{p*\cap\varphi*}]^{\varphi,x} \vdash_{\mathsf{M}} g : T^{q}\ \varepsilon_2 \\ \theta = [p/x] \quad x \notin \mathrm{fv}\,(T)\end{array}}{[\Sigma \mid \Gamma]^{\varphi} \vdash_{\mathsf{M}} \mathbf{let}\ x = b\ \mathbf{in}\ g : (T^{q}\ \varepsilon_1 \rhd \varepsilon_2)\theta}\ (\text{G-LET})$$

$$\frac{\begin{array}{c}\mathsf{x} : B^{\varnothing} \in [\Sigma \mid \Gamma]^{\varphi} \\ \mathsf{y} : \mathsf{Alloc}^{q} \in [\Sigma \mid \Gamma]^{\varphi}\end{array}}{[\Sigma \mid \Gamma]^{\varphi} \vdash_{\mathsf{M}} \mathbf{ref}_{\mathsf{y}}\ \mathsf{x} : (\mathsf{Ref}\ B)^{\varnothing}\ \mathsf{y}}\ (\text{N-REF}) \qquad \frac{\begin{array}{c}[\Sigma \mid \Gamma]^{\varphi} \vdash_{\mathsf{M}} b : S^{p}\ \varepsilon_1 \\ \Sigma \mid \Gamma \vdash S^{p}\ \varepsilon_1 <: T^{q}\ \varepsilon_2 \\ q, \varepsilon_2 \subseteq \varphi\end{array}}{[\Sigma \mid \Gamma]^{\varphi} \vdash_{\mathsf{M}} b : T^{q}\ \varepsilon_2}\ (\text{B-SUB})$$

$$\frac{\mathsf{x} : (\mathsf{Ref}\ B)^{q} \in [\Sigma \mid \Gamma]^{\varphi}}{[\Sigma \mid \Gamma]^{\varphi} \vdash_{\mathsf{M}} !\ \mathsf{x} : B^{\varnothing}\ \mathsf{x}}\ (\text{N-!})$$

Name Lookup $\boxed{\mathsf{x} : T^{q} \in [\Sigma \mid \Gamma]^{\varphi}}$

$$\frac{x : T^{q} \in \Gamma \qquad x \subseteq \varphi}{x : T^{q} \in [\Sigma \mid \Gamma]^{\varphi}}\ (\text{L-VAR}) \qquad \frac{\ell : T^{q} \in \Sigma \qquad \ell \subseteq \varphi}{\ell : T^{q} \in [\Sigma \mid \Gamma]^{\varphi}}\ (\text{L-LOC})$$

Fig. 6. The syntax and typing rules of the monadic normalform λ^*_{M}. Cf. Figure 2 for the subtyping rules.

In this work, we choose an even stricter form of MNF than usual, *i.e.*, sequences of let bindings in graph terms g always end with a name. We found that this more regular form is easier to work with when specifying optimization rules.

4.2 Reduction Preserves MNF

The reduction relation for λ^*_{ε} with store-allocated values (Figure 5) preserves MNF, and can thus be restricted to obtain the call-by-value reduction relation for λ^*_{M}:

Lemma 4.1 (Reduction Preserves MNF). *Let g be a graph term in λ^*_{M}, and σ a store that only binds λ^*_{M} introductions, such that $\sigma \mid g \longrightarrow_{\mathsf{sv}} \sigma' \mid t$ for some σ' and term t. Then it holds that*

(1) σ' is a store binding only λ^*_{M} introductions.
(2) t is a graph term of λ^*_{M}.

Proof. Since the reduction step begins with a graph term g, it can be only decomposed into a redex and evaluation context according to $E ::= \square \mid \mathbf{let}\ x = E\ \mathbf{in}\ g$, and no other cases apply.

$$\boxed{t \rightsquigarrow g}$$

$$\frac{}{c \rightsquigarrow \textbf{let } x = c \textbf{ in } x}$$

$$\frac{}{\mathsf{x} \rightsquigarrow \mathsf{x}}$$

$$\frac{t \rightsquigarrow g}{\lambda x.t \rightsquigarrow \textbf{let } y = \lambda x.g \textbf{ in } y}$$

$$\frac{t \rightsquigarrow g}{!t \rightsquigarrow \textbf{let } x_1 = g \textbf{ in let } x_2 = !x_1 \textbf{ in } x_2}$$

$$\frac{t_1 \rightsquigarrow g_1 \qquad t_2 \rightsquigarrow g_2}{\textbf{let } x = t_1 \textbf{ in } t_2 \rightsquigarrow \textbf{let } x = g_1 \textbf{ in } g_2}$$

$$\frac{t_1 \rightsquigarrow g_1 \qquad t_2 \rightsquigarrow g_2}{t_1\ t_2 \rightsquigarrow \textbf{let } x_1 = g_1 \textbf{ in let } x_2 = g_2 \textbf{ in let } x_3 = x_1\ x_2 \textbf{ in } x_3}$$

$$\frac{t_1 \rightsquigarrow g_1 \qquad t_2 \rightsquigarrow g_2}{\textbf{ref}_{t_1}\ t_2 \rightsquigarrow \textbf{let } x_1 = g_1 \textbf{ in let } x_2 = g_2 \textbf{ in let } x_3 = \textbf{ref}_{x_1}\ x_2 \textbf{ in } x_3}$$

$$\frac{t_1 \rightsquigarrow g_1 \qquad t_2 \rightsquigarrow g_2}{t_1 := t_2 \rightsquigarrow \textbf{let } x_1 = g_1 \textbf{ in let } x_2 = g_2 \textbf{ in let } x_3 = (x_1 := x_2) \textbf{ in } x_3}$$

Fig. 7. Translation from λ^*_ε into λ^*_{M}. Variable names introduced on the right-hand side are always fresh.

Redexes in the hole can only be bindings b, *i.e.*, either a node n or a nested graph term g'. It is easy to see that each possible reduction rule focuses on such a binding, and each rule plugs the hole with another binding b' on the right-hand side, thus preserving MNF. Furthermore, σ' is either equal to σ, or a modification of the latter where each binding is in MNF. □

4.3 Translation from Direct Style to MNF

This section considers the syntax-directed translation of λ^*_ε into λ^*_{M} (Figure 7).

Lemma 4.2 (Type Preservation of the MNF Translation). *If* $[\Sigma \mid \Gamma]^{\varphi} \vdash t : T^{q}\ \varepsilon$ *and* $t \rightsquigarrow g$*, then* $[\Sigma \mid \Gamma]^{\varphi} \vdash_{\mathsf{M}} g : T^{q}\ \varepsilon$*.*

Proof. Straightforward by induction over the typing derivation $[\Sigma \mid \Gamma]^{\varphi} \vdash t : T^{q}\ \varepsilon$. We exemplify the proof for applications (t-app). In this case

$$t_1\ t_2 \rightsquigarrow \textbf{let } x_1 = g_1 \textbf{ in let } x_2 = g_2 \textbf{ in let } x_3 = x_1\ x_2 \textbf{ in } x_3$$

where $t_1 \rightsquigarrow g_1$ and $t_2 \rightsquigarrow g_2$.

(1) We have $[\Sigma \mid \Gamma]^{\varphi} \vdash t_1 : \left(x : T^{p* \cap q*} \to^{\varepsilon_3} U^{r}\right)^{q}\ \varepsilon_1$.
(2) We have $[\Sigma \mid \Gamma]^{\varphi} \vdash t_2 : T^{p}\ \varepsilon_2$.
(3) We have $x \notin \mathrm{fv}(U)$, $\varepsilon_3 \subseteq q, x$, $r \subseteq \varphi, x$, and $\theta = [p/x]$.
(4) By IH: $[\Sigma \mid \Gamma]^{\varphi} \vdash_{\mathsf{M}} g_1 : \left(x : T^{p* \cap q*} \to^{\varepsilon_3} U^{r}\right)^{q}\ \varepsilon_1$.
(5) By IH: $[\Sigma \mid \Gamma]^{\varphi} \vdash_{\mathsf{M}} g_2 : T^{p}\ \varepsilon_2$.
(6) By weakening: $[\Sigma \mid \Gamma, x_1 : \left(x : T^{p* \cap q*} \to^{\varepsilon_3} U^{r}\right)^{q}]^{\varphi, x_1} \vdash_{\mathsf{M}} g_2 : T^{p}\ \varepsilon_2$.
(7) Let $\Gamma' := \Gamma, x_1 : \left(x : T^{p \cap q} \to^{\varepsilon_3} U^{r}\right)^{q}, x_2 : T^{p}$.
(8) By rule (n-app) and (3): $[\Sigma \mid \Gamma']^{\varphi, x_1, x_2} \vdash_{\mathsf{M}} x_1\ x_2 : U^{r\theta}\ \varepsilon_3\theta$.

(9) By (G-LET) and (G-RET): $[\Sigma \mid \Gamma']^{\varphi, x_1, x_2} \vdash_{\mathsf{M}} \textbf{let}\ x_3 = x_1\ x_2\ \textbf{in}\ x_3 : U^{r\theta}\ \varepsilon_3\theta$.

(10) With (6) and (G-LET):

$$[\Sigma \mid \Gamma, x_1 : \left(x : T^{p* \cap q*} \rightarrow^{\varepsilon_3} U^{r}\right)^{q}]^{\varphi, x_1} \vdash_{\mathsf{M}} \textbf{let}\ x_2 = g_2\ \textbf{in}\ \textbf{let}\ x_3 = x_1\ x_2\ \textbf{in}\ x_3 : (U^{r\theta}\ \varepsilon_2 \rhd \varepsilon_3\theta)[p/x_2]$$

(11) With (4) and (G-LET):

$$[\Sigma \mid \Gamma]^{\varphi} \vdash_{\mathsf{M}} \textbf{let}\ x_1 = g_1\ \textbf{in}\ \textbf{let}\ x_2 = g_2\ \textbf{in}\ \textbf{let}\ x_3 = x_1\ x_2\ \textbf{in}\ x_3 : (U^{r\theta[p/x_2]}\ \varepsilon_1 \rhd (\varepsilon_2 \rhd \varepsilon_3\theta)[p/x_2])[q/x_1]$$

(12) Since x_1 and x_2 were picked fresh, and x is not free in ε_1 and ε_2 by (1) and (2), we have

$$(U^{r\theta[p/x_2]}\ \varepsilon_1 \rhd (\varepsilon_2 \rhd \varepsilon_3\theta)[p/x_2])[q/x_1] = U^{r\theta}\ \varepsilon_1 \rhd \varepsilon_2 \rhd \varepsilon_3\theta = (U^{r}\ \varepsilon_1 \rhd \varepsilon_2 \rhd \varepsilon_3)\theta.$$

That is, (11) proves the goal.

□

4.4 Soundness

Instead of proving progress and preservation directly, we assert that terms in monadic normal form can always be typed in the same manner in both the direct style and MNF type systems. The intention is that we have the same type system, but restricted in the terms.

Lemma 4.3 (Type-preserving Embedding of MNF Terms).

(1) $[\Sigma \mid \Gamma]^{\varphi} \vdash_{\mathsf{M}} n : T^{q}\ \varepsilon$ *iff* $[\Sigma \mid \Gamma]^{\varphi} \vdash n : T^{q}\ \varepsilon$.

(2) $[\Sigma \mid \Gamma]^{\varphi} \vdash_{\mathsf{M}} g : T^{q}\ \varepsilon$ *iff* $[\Sigma \mid \Gamma]^{\varphi} \vdash g : T^{q}\ \varepsilon$.

Proof. Each direction is proved by mutual induction over the respective typing derivation. □

Together with Lemma 4.1, it follows that the type soundness and preservation of separation results of the direct style system (Section 3.3) carry over to the MNF version.

Corollary 4.4 (MNF Progress). *If* $[\Sigma \mid \varnothing]^{\mathrm{dom}(\Sigma)} \vdash_{\mathsf{M}} g : T^{q}\ \varepsilon$, *then either* g *is a value, or for any store* σ *where* $[\Sigma \mid \varnothing]^{\mathrm{dom}(\Sigma)} \vdash \sigma$, *there exists a graph term* g' *and store* σ' *such that* $\sigma \mid g \longrightarrow_{\mathbf{sv}} \sigma' \mid g'$.

Corollary 4.5 (MNF Preservation).

$$\frac{[\Sigma \mid \varnothing]^{\mathrm{dom}(\Sigma)} \vdash_{\mathsf{M}} g : T^{q}\ \varepsilon \qquad [\Sigma \mid \varnothing]^{\mathrm{dom}(\Sigma)} \vdash \sigma \qquad \sigma \mid g \longrightarrow_{\mathbf{sv}} \sigma' \mid g'}{\exists \Sigma' \supseteq \Sigma.\ \exists p \subseteq \mathrm{dom}(\Sigma' \setminus \Sigma). \quad [\Sigma' \mid \varnothing]^{\mathrm{dom}(\Sigma')} \vdash_{\mathsf{M}} g' : T^{q,p}\ \varepsilon, p \qquad [\Sigma' \mid \varnothing]^{\mathrm{dom}(\Sigma')} \vdash \sigma'}$$

Corollary 4.6 (MNF Preservation of Separation). *Interleaved executions preserve types and disjointness:*

$$\frac{\begin{array}{lll} [\Sigma \mid \varnothing]^{\mathrm{dom}(\Sigma)} \vdash_{\mathsf{M}} g_1 : T_1^{q_1}\ \varepsilon_1 & \sigma\ \mid g_1 \longrightarrow_{\mathbf{sv}} \sigma'\ \mid g_1' & [\Sigma \mid \varnothing]^{\mathrm{dom}(\Sigma)} \vdash \sigma \\ [\Sigma \mid \varnothing]^{\mathrm{dom}(\Sigma)} \vdash_{\mathsf{M}} g_2 : T_2^{q_2}\ \varepsilon_2 & \sigma' \mid g_2 \longrightarrow_{\mathbf{sv}} \sigma'' \mid g_2' & q_1 \cap q_2 \subseteq \varnothing \end{array}}{\begin{array}{lll} \exists p_1\ p_2\ \varepsilon_1'\ \varepsilon_2'\ \Sigma'\ \Sigma''. & [\Sigma'\ \mid \varnothing]^{\mathrm{dom}(\Sigma')} \vdash_{\mathsf{M}} g_1' : T_1^{p_1}\ \varepsilon_1' & \Sigma'' \supseteq \Sigma' \supseteq \Sigma \\ & [\Sigma'' \mid \varnothing]^{\mathrm{dom}(\Sigma'')} \vdash_{\mathsf{M}} g_2' : T_2^{p_2}\ \varepsilon_2' & p_1 \cap p_2 \subseteq \varnothing \end{array}}$$

5 MONADIC NORMAL FORM WITH HARD DEPENDENCIES

This section presents our graph IR λ^*_{G} which enriches the monadic normal form (MNF) of the previous section with effect dependencies.

Graph IR $\boxed{\lambda^{*}_{\mathrm{G}}}$

$$\begin{array}{llll}
\mathsf{x}, \mathsf{y}, \mathsf{z} & ::= & x \mid \ell & \text{Names} \\
\iota & ::= & c \mid \lambda x.(g \bullet \delta) \mid \mathbf{ref}_{\ell}\ \ell & \text{Introductions} \\
v & ::= & \ell & \text{Values} \\
n & ::= & \iota \mid \mathsf{x}\ \mathsf{x} \mid \mathbf{ref}_{\mathsf{x}}\ \mathsf{x} \mid !\ \mathsf{x} \mid \mathsf{x} := \mathsf{x} & \text{Graph Nodes} \\
g & ::= & \mathsf{x} \mid \mathbf{let}\ x = b \bullet \delta\ \mathbf{in}\ g & \text{Graph Terms} \\
b & ::= & n \mid g & \text{Bindings} \\
\Delta, \delta & ::= & \overline{\mathsf{x} \mapsto \mathsf{x}} & \text{Dependencies}
\end{array}$$

Typing $\boxed{[\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash (n \mid g) : T^{q}\ \varepsilon}$

$$\frac{c \in B}{[\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash c : B^{\varnothing}\ \varnothing}\ (\text{N-CST})
\qquad
\frac{\mathsf{x} : (\mathsf{Ref}\ B)^{q} \in [\Sigma \mid \Gamma]^{\varphi} \quad \mathsf{y} : B^{\varnothing} \in [\Sigma \mid \Gamma]^{\varphi}}{[\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash \mathsf{x} := \mathsf{y} : \mathsf{Unit}^{\varnothing}\ \mathsf{x}}\ (\text{N-:=})$$

$$\frac{[\Sigma \mid \Gamma, x : T^{p}]^{q,x} \bullet \mapsto x \vdash g : U^{r}\ \varepsilon \qquad q \subseteq \varphi \qquad \delta \sqsubseteq \varepsilon{*} \mapsto x}{[\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash \lambda x.g \bullet \delta : (x : T^{p} \to^{\varepsilon} U^{r})^{q}\ \varnothing}\ (\text{N-ABS})
\qquad
\frac{\mathsf{x} : T^{q} \in [\Sigma \mid \Gamma]^{\varphi}}{[\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash \mathsf{x} : T^{\mathsf{x}}\ \varnothing}\ (\text{G-RET})$$

$$\frac{\mathsf{x} : \left(z : T^{p{*}\cap q{*}} \to^{\varepsilon} U^{r}\right)^{q} \in [\Sigma \mid \Gamma]^{\varphi} \quad \mathsf{y} : T^{p} \in [\Sigma \mid \Gamma]^{\varphi} \quad \theta = [p/z] \quad z \notin \mathrm{fv}(U) \quad \varepsilon \subseteq q, z \quad r \subseteq \varphi, z}{[\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash \mathsf{x}\ \mathsf{y} : (U^{r}\ \varepsilon)\theta}\ (\text{N-APP})$$

$$\frac{[\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash b : S^{p}\ \varepsilon_1 \quad [\Sigma \mid \Gamma, x : S^{p{*}\cap\varphi{*}}]^{\varphi,x} \bullet \Delta, (\varepsilon_1{*}, x) \mapsto x \vdash g : T^{q}\ \varepsilon_2 \quad \theta = [p/x] \quad x \notin \mathrm{fv}\,(T) \quad \delta \sqsubseteq \Delta|_{\varepsilon_1{*}}}{[\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash \mathbf{let}\ x = b \bullet \delta\ \mathbf{in}\ g : (T^{q}\ \varepsilon_1 \rhd \varepsilon_2)\theta}\ (\text{G-LET})$$

$$\frac{\mathsf{x} : \mathsf{Alloc}^{q} \in [\Sigma \mid \Gamma]^{\varphi} \quad \mathsf{y} : B^{\varnothing} \in [\Sigma \mid \Gamma]^{\varphi}}{[\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash \mathbf{ref}_{\mathsf{x}}\ \mathsf{y} : (\mathsf{Ref}\ B)^{\varnothing}\ \mathsf{x}}\ (\text{N-REF})$$

$$\frac{\mathsf{x} : (\mathsf{Ref}\ B)^{q} \in [\Sigma \mid \Gamma]^{\varphi}}{[\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash !\ \mathsf{x} : B^{\varnothing}\ \mathsf{x}}\ (\text{N-!})
\qquad
\frac{[\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash b : S^{p}\ \varepsilon_1 \quad \Sigma \mid \Gamma \vdash S^{p}\ \varepsilon_1 <: T^{q}\ \varepsilon_2 \quad q, \varepsilon_2 \subseteq \varphi}{[\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash b : T^{q}\ \varepsilon_2}\ (\text{B-SUB})$$

Name Lookup (cf. Figure 6) $\boxed{\mathsf{x} \in [\Sigma \mid \Gamma]^{\varphi}}$

Well-Typed Runtime Configuration $\boxed{[\Sigma \mid \Gamma]^{\varphi} \vdash z.g \bullet \delta : T^{q}\ \varepsilon}$

$$\frac{[\Sigma \mid \Gamma]^{\varphi} \bullet \mapsto z \vdash g : T^{q}\ \varepsilon \qquad \delta \sqsubseteq \varepsilon \mapsto z \qquad q = q{*} \qquad \varepsilon = \varepsilon{*}}{[\Sigma \mid \Gamma]^{\varphi} \vdash z.g \bullet \delta : T^{q}\ \varepsilon}$$

Well-formed Contexts $\boxed{\Sigma \mid \Gamma \bullet \Delta.z\ \mathrm{ok}}$

$$\frac{\mathrm{dom}(\Delta) \subseteq \mathrm{dom}(\Sigma), \mathrm{dom}(\Gamma) \qquad \mathrm{cod}(\Delta) \subseteq \mathrm{dom}(\Sigma), \mathrm{dom}(\Gamma), z}{\Sigma \mid \Gamma \bullet \Delta.z\ \mathrm{ok}}$$

Fig. 8. The syntax and typing rules of the graph IR λ^{*}_{G}. Cf. Figure 2 for the subtyping rules.

5.1 Dependencies

Data dependencies are expressed by ordinary variable occurrences in terms. Tracking effect dependencies requires extra term annotations. Intuitively, a (hard) effect dependency $x \mapsto y$ indicates that an effect on the node x (*e.g.*, a reference, or global capability) is induced, and that node y is the

Dependency Synthesis

$$\boxed{[\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash (n \mid g) : T^{q}\ \boldsymbol{\varepsilon} \rightsquigarrow (\boldsymbol{n} \mid \boldsymbol{g}) \bullet \delta}$$

$$\frac{c \in B}{[\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash c : B^{\varnothing}\ \boldsymbol{\varnothing} \rightsquigarrow c \bullet \varnothing} \quad (\rightsquigarrow\text{-CST})$$

$$\frac{\begin{array}{c}[\Sigma \mid \Gamma, x : T^{p}]^{q,x} \bullet \mapsto x \vdash g : U^{r}\ \boldsymbol{\varepsilon} \rightsquigarrow \boldsymbol{g} \bullet \delta \\ q \subseteq \varphi\end{array}}{\begin{array}{c}[\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash \lambda x.g : (x : T^{p} \to^{\varepsilon} U^{r})^{q}\ \boldsymbol{\varnothing} \\ \rightsquigarrow (\lambda x.\boldsymbol{g} \bullet \delta) \bullet \varnothing\end{array}} \quad (\rightsquigarrow\text{-ABS})$$

$$\frac{\begin{array}{c}\mathsf{x} : \left(z : T^{p*\cap q*} \to^{\varepsilon} U^{r}\right)^{q} \in [\Sigma \mid \Gamma]^{\varphi} \\ \mathsf{y} : T^{p} \in [\Sigma \mid \Gamma]^{\varphi} \quad \theta = [p/z] \\ z \notin \mathrm{fv}(U) \quad \boldsymbol{\varepsilon} \subseteq q, z \quad r \subseteq \varphi, z\end{array}}{\begin{array}{c}[\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash \mathsf{x}\ \mathsf{y} : (U^{r}\ \boldsymbol{\varepsilon})\theta \\ \rightsquigarrow \mathsf{x}\ \mathsf{y} \bullet \Delta|_{(\varepsilon\theta)*}\end{array}} \quad (\rightsquigarrow\text{-APP})$$

$$\frac{\begin{array}{c}\mathsf{x} : \mathsf{Alloc}^{q} \in [\Sigma \mid \Gamma]^{\varphi} \\ \mathsf{y} : B^{\varnothing} \in [\Sigma \mid \Gamma]^{\varphi}\end{array}}{\begin{array}{c}[\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash \mathbf{ref}_{\mathsf{x}}\ \mathsf{y} : (\mathsf{Ref}\ B)^{\varnothing}\ \mathsf{x} \\ \rightsquigarrow \mathbf{ref}_{\mathsf{x}}\ \mathsf{y} \bullet \Delta|_{\mathsf{x}*}\end{array}} \quad (\rightsquigarrow\text{-REF})$$

$$\frac{\mathsf{x} : (\mathsf{Ref}\ B)^{q} \in [\Sigma \mid \Gamma]^{\varphi}}{\begin{array}{c}[\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash\ !\,\mathsf{x} : B^{\varnothing}\ \mathsf{x} \\ \rightsquigarrow\ !\,\mathsf{x} \bullet \Delta|_{\mathsf{x}*}\end{array}} \quad (\rightsquigarrow\text{-!})$$

$$\frac{\begin{array}{c}\mathsf{x} : (\mathsf{Ref}\ B)^{q} \in [\Sigma \mid \Gamma]^{\varphi} \\ \mathsf{y} : B^{\varnothing} \in [\Sigma \mid \Gamma]^{\varphi}\end{array}}{\begin{array}{c}[\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash \mathsf{x} := \mathsf{y} : \mathsf{Unit}^{\varnothing}\ \mathsf{x} \\ \rightsquigarrow \mathsf{x} := \mathsf{y} \bullet \Delta|_{\mathsf{x}*}\end{array}} \quad (\rightsquigarrow\text{-:=})$$

$$\frac{\mathsf{x} : T^{q} \in [\Sigma \mid \Gamma]^{\varphi}}{[\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash \mathsf{x} : T^{\mathsf{x}}\ \boldsymbol{\varnothing} \rightsquigarrow \mathsf{x} \bullet \varnothing} \quad (\rightsquigarrow\text{-RET})$$

$$\frac{\begin{array}{c}[\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash b : S^{p}\ \boldsymbol{\varepsilon_1} \rightsquigarrow \boldsymbol{b} \bullet \delta_1 \\ [\Sigma \mid \Gamma, x : S^{p*\cap\varphi*}]^{\varphi,x} \bullet \Delta, (\boldsymbol{\varepsilon_1}*, x) \mapsto x \vdash g : T^{q}\ \boldsymbol{\varepsilon_2} \\ \rightsquigarrow \boldsymbol{g} \bullet \delta_2 \\ \theta = [p/x] \quad x \notin \mathrm{fv}\,(T)\end{array}}{\begin{array}{c}[\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash \mathbf{let}\ x = b\ \mathbf{in}\ g : (T^{q}\ \boldsymbol{\varepsilon_1} \rhd \boldsymbol{\varepsilon_2})\theta \\ \rightsquigarrow (\mathbf{let}\ x = \boldsymbol{b} \bullet \delta_1\ \mathbf{in}\ \boldsymbol{g}) \bullet \delta_1, \delta_2[x \rightsquigarrow \Delta|_{p*}]\end{array}} \quad (\rightsquigarrow\text{-LET})$$

$$\frac{\begin{array}{c}[\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash b : S^{p}\ \boldsymbol{\varepsilon_1} \rightsquigarrow \boldsymbol{b} \bullet \delta_1 \\ \Sigma \mid \Gamma \vdash S^{p}\ \boldsymbol{\varepsilon_1} <: T^{q}\ \boldsymbol{\varepsilon_2} \\ q, \boldsymbol{\varepsilon_2} \subseteq \varphi\end{array}}{[\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash b : T^{q}\ \boldsymbol{\varepsilon_2} \rightsquigarrow \boldsymbol{b} \bullet \Delta|_{\varepsilon_2*}} \quad (\rightsquigarrow\text{-SUB})$$

Name Lookup (cf. Figure 6)

$$\boxed{\mathsf{x} \in [\Sigma \mid \Gamma]^{\varphi}}$$

Fig. 9. Type-and-effect-directed dependency synthesis from λ^{*}_{M} into λ^{*}_{G}.

previous node in the graph at which an effect on x occurred. That is to say, $x \mapsto y$ *does not* indicate there is an edge between x and y, but rather that there is an edge *from the current node* to which $x \mapsto y$ is attached pointing to y and its (effect) label is x. Oftentimes we will just say "dependency" and omit the "hard" and "effect" qualifiers.

We bundle these dependencies into finite maps δ from variables to variables, and annotate them to atomic nodes or nested graphs at let bindings. Dependencies come with a standard "update" operator which is associative:

$$(\delta_1, \delta_2)(x) := \begin{cases} \delta_2(x) & x \in \mathrm{dom}(\delta_2) \\ \delta_1(x) & \text{otherwise.} \end{cases}$$

and a restriction operator of the domain to a given set (abusing set notation):

$$\delta|_{\alpha} := \{x \mapsto y \in \delta \mid x \in \alpha\},$$

and we also define two removal operators, i.e.,

$$\delta - \alpha := \{x \mapsto y \in \delta \mid y \notin \alpha\},$$

which removes all mappings pointing into α, and

$$\delta \setminus \alpha := \delta|_{\mathrm{dom}(\delta)\setminus\alpha},$$

Store Terms, Graph Term Contexts, Binding Contexts

$$\begin{array}{rcl} \sigma & ::= & \varnothing \mid \sigma,\ \mathbf{let}_{\mathrm{s}}\ \ell = \iota \\ G & ::= & \square \bullet \delta \mid (\mathbf{let}\ x = G\ \mathbf{in}\ g) \bullet \delta \\ B & ::= & (\mathbf{let}\ x = \square\ \mathbf{in}\ g) \bullet \delta \mid (\mathbf{let}\ x = B\ \mathbf{in}\ g) \bullet \delta \end{array}$$

Reduction Rules $\boxed{\sigma \mid z.g \bullet \delta \longrightarrow_{\mathrm{G}} \sigma \mid z.g \bullet \delta}$

$$\begin{array}{rcll} \sigma \mid z.B[\ \ell_1\ \ell_2 \bullet \delta_1\ ] & \longrightarrow_{\mathrm{G}} & \sigma \mid z.B[\ (g \bullet \delta_2)[x \rightsquigarrow \delta_1][\ell_2/x][\ell_2/x]_{\mathrm{t}}\ ] & (\beta) \\ & & \sigma = \sigma_1,\ \mathbf{let}_{\mathrm{s}}\ \ell_1 = \lambda x.g \bullet \delta_2,\ \sigma_2 & \\ & & \mathrm{dom}(\delta_1) \subseteq \mathrm{dom}(\sigma),\ \mathrm{cod}(\delta_1) \subseteq \{z\} & \\ \sigma \mid z.G[\ \mathbf{let}\ x = \ell \bullet \delta\ \mathbf{in}\ g\ ] & \longrightarrow_{\mathrm{G}} & \sigma \mid z.G[\ g[x \rightsquigarrow \delta][\ell/x][\ell/x]_{\mathrm{t}}\ ] & (\textsc{let}) \\ & & \mathrm{dom}(\delta) \subseteq \mathrm{dom}(\sigma),\ \mathrm{cod}(\delta) \subseteq \{z\} & \\ \sigma \mid z.G[\ \mathbf{let}\ x = \iota \bullet \delta\ \mathbf{in}\ g\ ] & \longrightarrow_{\mathrm{G}} & \sigma, \mathbf{let}_{\mathrm{s}}\ \ell = \iota \mid z.G'[\ g[x \rightsquigarrow \delta][\ell/x][\ell/x]_{\mathrm{t}}\ ] & (\textsc{intro}) \\ & & \ell \notin \mathrm{dom}(\sigma),\ \mathrm{cod}(\delta) \subseteq \{z\},\ G' = G\langle \iota : z.\ell \rangle & \\ \sigma,\ \mathbf{let}_{\mathrm{s}}\ \ell = \mathbf{ref}_w\ \ell',\ \sigma' \mid z.B[\ !\ell \bullet \delta\ ] & \longrightarrow_{\mathrm{G}} & \sigma,\ \mathbf{let}_{\mathrm{s}}\ \ell = \mathbf{ref}_w\ \ell',\ \sigma' \mid z.B[\ \ell' \bullet \delta\ ] & (\textsc{deref}) \\ & & \mathrm{dom}(\delta) \subseteq \mathrm{dom}(\sigma),\ \mathrm{cod}(\delta) \subseteq \{z\} & \\ \sigma,\ \mathbf{let}_{\mathrm{s}}\ \ell = \mathbf{ref}_w\ \ell',\ \sigma' \mid z.B[\ \ell := \ell'' \bullet \delta\ ] & \longrightarrow_{\mathrm{G}} & \sigma,\ \mathbf{let}_{\mathrm{s}}\ \ell = \mathbf{ref}_w\ \ell'',\ \sigma' \mid z.B[\ \mathrm{unit} \bullet \delta\ ] & (\textsc{assign}) \\ & & \mathrm{dom}(\delta) \subseteq \mathrm{dom}(\sigma),\ \mathrm{cod}(\delta) \subseteq \{z\} & \end{array}$$

Contextual Effect Propagation $\boxed{G\langle \iota : z.\ell \rangle}$

$$\begin{array}{lcl} G\langle c : z.\ell \rangle & = & G \\ G\langle \lambda x.g \bullet \delta : z.\ell \rangle & = & G \\ (\square \bullet \delta)\langle \mathbf{ref}_w\ \ell_1 : z.\ell_2 \rangle & = & \square \bullet \delta, \ell_2 \mapsto z \\ ((\mathbf{let}\ x = G\ \mathbf{in}\ g) \bullet \delta)\langle \mathbf{ref}_w\ \ell_1 : z.\ell_2 \rangle & = & (\mathbf{let}\ x = G\langle \mathbf{ref}_w\ \ell_1 : z.\ell_2 \rangle\ \mathbf{in}\ g) \bullet \delta, \ell_2 \mapsto z \end{array}$$

Well-Formed Store Entries and Stores $\boxed{[\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash \ell : \iota \in \sigma}$ $\boxed{[\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash \sigma}$

$$\frac{\Sigma(\ell) = \mathrm{Ref}\ B^{\varnothing} \qquad [\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash \ell' : B^{\varnothing}\ \varnothing \qquad \Sigma(w) = \mathrm{Alloc}^{\varnothing} \qquad \sigma(w) = \omega}{[\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash \ell : \mathbf{ref}_w\ \ell' \in \sigma}$$

$$\frac{\Sigma(\ell) = T^{q} \qquad [\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash \iota : T^{q}\ \varnothing \qquad \forall \ell, w.\ \iota \neq \mathbf{ref}_w\ \ell}{[\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash \ell : \iota \in \sigma}$$

$$\frac{|\Sigma| = |\sigma| \qquad \left([\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash \ell : \iota \in \sigma\right)_{\mathbf{let}_{\mathrm{s}}\ \ell = \iota \in \sigma}}{[\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash \sigma}$$

Fig. 10. Call-by-value reduction for λ^{*}_{G} with runtime dependency checking.

which removes all entries pointing from α. In symmetry with substitutions on terms and qualifiers, there is a notion of substitution (or rewiring, rerouting), over dependencies:

$$\delta_1[x \rightsquigarrow \delta_2] := \delta_1 - \{x\}, \{y \mapsto z \in \delta_2 \mid y \mapsto x \in \delta_1\},$$

which is rerouting the dependency target x via δ_2. For instance,

$$(a \mapsto b, c \mapsto x, y \mapsto x)[x \rightsquigarrow (c \mapsto a, y \mapsto b)] = (a \mapsto b, c \mapsto a, y \mapsto b).$$

That is, rewiring substitutes some of the targets in dependency mappings, which happens when the variable x is replaced, e.g., when x is the formal parameter of a function at a call site. The function body may have dependencies pointing to x, and these must be rerouted using the call-site's dependencies. Dually, x might also be in the domain of a dependency, and the monadic semantics will rename it to some store location holding the value of the call site argument. For this case, we also lift qualifier substitutions to act on the domain of dependencies, i.e.,

$$\delta[q/x] := \delta \setminus x, \{y \mapsto \delta(x) \mid y \in q\}$$

so that all variables and locations in q will point to x's target, if it has an entry.

Effect-dependency calculation. Static effects ε and contextual information on last uses determine effect dependencies δ at graph nodes. We outline the basic process in the following.

A few auxiliary definitions are in order:

$$\begin{aligned}
\alpha \mapsto x &:= \{y \mapsto x \mid y \in \alpha\} \\
\Sigma, \Gamma \mapsto x &:= \mathrm{dom}(\Sigma) \cup \mathrm{dom}(\Gamma) \mapsto x \\
\Sigma \mid \Gamma \vdash \mapsto x &:= \Sigma, \Gamma \mapsto x,\ \ \Sigma, \Gamma \text{ are implicit} \\
\Sigma \mid \Gamma \vdash \delta \uparrow^{z} &:= \delta, \{x \mapsto z \mid x \in (\mathrm{dom}(\Sigma) \cup \mathrm{dom}(\Gamma)) \setminus \mathrm{dom}(\delta)\},\ \ \Sigma, \Gamma \text{ are implicit}
\end{aligned}$$

Firstly, we overload the finite maps notation to specify a map with domain α pointing to the single variable x, and analogously the map pointing all the variables and locations bound in Γ and Σ to x, for which we often use the shorthand notation $\mapsto x$ omitting those contexts when they are unambiguous. The $\uparrow^{z}$ operator is used to ensure that a dependency is defined for at least the variables and location in context, adding entries pointing to a common block start variable z.

If a nested graph named x has effect ε, then we record that the reachable variables affected by $\varepsilon*$[5] were last used at x (plus x was last used at itself, explained below). So if Δ records the last use of each variable in scope (essentially a structural coeffect which is also just a finite map from variables to variables), then we update it to $\Delta, (\varepsilon*, x \mapsto x)$. Synthesizing the dependency for node x itself is just a matter of projecting the last-use coeffect Δ.

Concretely, consider **let** $x = g_1$ **in** g_2 with last uses Δ, and let ε be the effect of g_1. We first calculate the annotated version of g_1:

$$g_1 \rightsquigarrow g_1' \bullet \Delta|_{\varepsilon*}$$

which attaches the current last uses with respect to ε. When proceeding with the continuation g_2, x is added into the context, and we need to define its last-use coeffect, which is simply x itself. We will discuss the precise calculation rules of our type system in the next section.

5.2 Syntax and Statics

Figure 8 shows the type checking relation, and it is easy too see that the λ^{*}_{G}-calculus is identical to the λ^{*}_{M}-calculus (Figure 6), when erasing all the teal parts pertaining to dependencies. At the term level, we attach dependencies δ to let bindings (representing the dependencies for all the effects of the bound graph term) and the body of λ-abstractions (representing the dependencies of the abstraction's latent effects)

5.2.1 Type Checking. The typing judgment now carries an additional dependency map Δ attached to the context (basically a form of coeffect), which is used to track the *last uses* of variables and locations in the context/store. We stipulate that at all times the domain of Δ ranges over the domain of Γ and Σ. Last uses are threaded as an input through typing derivations, and the only rules at which they are accessed are those for terms with dependency annotations, *i.e.*, (N-ABS) and (G-LET). Those annotated dependencies should always conform to the effect of the term in question, and

[5] Since the systems in this report *lazily* assign qualifiers and effects, we need to consider the transitive reachability closure (Figure 3) to get ahold of all relevant dependencies.

as a rule of thumb, we regard the effect ε's transitive reachability closure (a set of variables and locations) as a slice of the currently known last uses Δ, *i.e.*, restricting the domain of Δ to the effect in question yields all the relevant dependencies at the current node.

In the rule (N-ABS) for λ-abstractions, we check the body with all the last uses pointing to the formal parameter x. This is because we generally do not know the call site and actual argument in advance, and the most natural choice is abstracting the last use of the free variables in the body by x in symmetry to term abstraction. As we have motivated in the main paper and will shortly see in the operational semantics (Section 5.3), the "latent" dependency δ annotated to the function's body has to be rewired at the call site. Following the "rule of thumb" above, we check that the annotated dependency δ is conforming to the body's dependency in relation to its last uses, *i.e.*, δ is a sub-map of $\varepsilon * \mapsto x$.

Similarly, rule (G-LET) checks that the annotated dependency for the bound node/nested graph is conforming to its effect, *i.e.*, it is a sub-map of the last uses for $\varepsilon_1 *$ in Δ. When typing the let body g, we update the last uses for the variables affected by $\varepsilon_1 *$ and let them point to the binding x, precisely because those variables have been last used here. Furthermore, since x is newly introduced in the body g, we also have to specify a last use for it, which is x itself.

5.2.2 Dependency Synthesis. While the typing relation provides a means to check dependencies, it does not provide a method to compute them. For this purpose we define a type/qualifier/effect-directed synthesis relation (Figure 9) for λ^*_{M} terms which lack any dependency annotations and produces dependency-annotated λ^*_{G} terms along with the dependency map for effects on free variables as output, given an initial map Δ of last uses and typing context as input. Synthesizing the dependencies follows the "rule of thumb" from above, *i.e.*, synthesized and inserted dependencies are always the currently known last uses of the variables in the term's effect (cf. Section 5.4.1).

5.3 Dynamics

The call-by-value operational semantics for λ^*_{G} (Figure 10) is a refinement of the operational semantics for λ^*_{M} with store-allocated values (Figure 5). The changes are twofold: (1) dependencies are part of the term syntax now, and have to be accounted for by reductions, and (2) the semantics additionally checks for each reduction step whether all the effect dependencies have been already evaluated and committed to the store. By type soundness (cf. Section 5.4), well-typed terms do not exhibit dependency violations, *i.e.*, dependencies correctly reflect the observed runtime execution order of effect operations.

Reduction occurs over runtime configurations $\sigma \mid z.g \bullet \delta$, which compared to the λ^*_{M}-calculus attaches a distinguished *start variable* z to the graph term g along with its dependency δ. We stipulate that z is always chosen so that it is not a free variable of g. It is a mechanism to check whether a dependency has already been evaluated. That is, at the top level, we set the initial use of all the free variables/locations to z, and that will be reflected in the synthesized dependencies of a term. The invariant is that the next operation will have all its dependencies purely on store variables and these will point to z, which is outside of the program. The meaning is that all dependencies of the current node are in the store, and each of the reduction rules checks this property. In the following, we discuss the changes made to the reduction rules compared to Figure 5.

In the function application rule (β), we now have to account for the latent dependencies of the function and the dependencies at the call site. Thus, in symmetry with dependent function application, we rewire the function body and its dependency δ_2 with δ_1 for x. Since dependencies are type-level information annotated in the term syntax, we also have to perform the qualifier substitution part of the static dependent function application, *i.e.*, we substitute the argument's location ℓ_2 for x in the domain of dependencies, if present. To distinguish qualifier substitution in dependencies from term substitution/renaming of variables, we attach the subscript t to the latter. Hence, substitution becomes simultaneous rewiring, qualifier substitution, and renaming on terms in the graph IR. Accordingly, we do the same in (LET) and (INTRO).

Rule (INTRO) has changed compared to Figure 5 in that it simultaneously performs the (LET) elimination along with the introduction of the fresh location. The reason is that from the perspective of static typing, introducing the new location will make it appear in reachability qualifiers and effects, and consequently in dependencies. So the reduction step has to patch up the dependencies in the body of the let expression, because x now aliases ℓ, which it did not beforehand. Furthermore, if ι is a mutable reference, using it makes it also appear in the codomain of effects dependencies, and this change needs propagating into all dependencies along the spine of the evaluation context up to the top level, because ℓ is globally visible.

Finally, modulo checking effect dependencies, rules (DEREF) and (ASSIGN) have not changed.

5.4 Metatheory

5.4.1 Properties of Dependency Synthesis. Dependency synthesis induces a function over MNF typing derivations which given an input map of last uses always produce an annotated graph IR term with the same type, qualifier, and effect. As a corollary, we obtain a type/effect/qualifier-preserving and dependency-synthesizing translation from the direct style λ^*_ε system into the λ^*_{G} graph IR.

LEMMA 5.1 (SYNTHESIS INVARIANT). *Dependencies are completely determined by the context and effect, as follows:*

(1) If $[\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash n : T^{q}\ \boldsymbol{\varepsilon} \rightsquigarrow \boldsymbol{n} \bullet \delta$, *then* $\delta = \Delta|_{\boldsymbol{\varepsilon}*}$.
(2) If $[\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash g : T^{q}\ \boldsymbol{\varepsilon} \rightsquigarrow \boldsymbol{g} \bullet \delta$, *then* $\delta = \Delta|_{\boldsymbol{\varepsilon}*}$.

PROOF. By mutual induction over the respective derivation. Most cases are straightforward. The case for rule ($\rightsquigarrow$-LET) requires careful reasoning about dependency maps:

- Case ($\rightsquigarrow$-LET): We need to show $\delta_1, \delta_2[x \rightsquigarrow \Delta|_p] = \Delta|_{(\boldsymbol{\varepsilon}_1 \triangleright \boldsymbol{\varepsilon}_2\theta)*}$ for $\theta = [p/x]$.
  (1) By IH: $\delta_1 = \Delta|_{\boldsymbol{\varepsilon}_1*}$.
  (2) By IH: $\delta_2 = (\Delta, (\boldsymbol{\varepsilon}_1*, x) \mapsto x)|_{\boldsymbol{\varepsilon}_2*}$.
  (3) By scoping: $x \notin \mathrm{dom}(\Delta)$, and $x \notin \mathrm{cod}(\Delta)$.
  (4) Case distinction:
    (a) $x \notin \boldsymbol{\varepsilon}_2$: Thus,

$$\begin{array}{rcll}
\delta_1, \delta_2[x \rightsquigarrow \Delta|_p] & = & \delta_1, \delta_2 & \text{by } x \notin \boldsymbol{\varepsilon}_2 \\
 & = & \Delta|_{\boldsymbol{\varepsilon}_1*}, (\Delta, (\boldsymbol{\varepsilon}_1*, x) \mapsto x)|_{\boldsymbol{\varepsilon}_2*} & \text{by (1), (2)} \\
 & = & \Delta|_{\boldsymbol{\varepsilon}_1*}, \Delta|_{\boldsymbol{\varepsilon}_2* \setminus \boldsymbol{\varepsilon}_1*} & \text{by } x \notin \boldsymbol{\varepsilon}_2 \text{ and def. of } \_, \_ \\
 & = & \Delta|_{\boldsymbol{\varepsilon}_1*, \boldsymbol{\varepsilon}_2*} & \text{by definition of } \_|\_ \\
 & = & \Delta|_{\boldsymbol{\varepsilon}_1* \triangleright \boldsymbol{\varepsilon}_2*} & \text{by definition of } \_ \triangleright \_ \\
 & = & \Delta|_{(\boldsymbol{\varepsilon}_1 \triangleright \boldsymbol{\varepsilon}_2)*} & \text{by properties of } \_* \\
 & = & \Delta|_{(\boldsymbol{\varepsilon}_1 \triangleright \boldsymbol{\varepsilon}_2\theta)*} \quad \checkmark & \text{by } x \notin \boldsymbol{\varepsilon}_2
\end{array}$$

    (b) $x \in \boldsymbol{\varepsilon}_2$:
      (i) We have that

$$\begin{array}{rcll}
\delta_2 & = & (\Delta, (\boldsymbol{\varepsilon}_1*, x) \mapsto x)|_{\boldsymbol{\varepsilon}_2*} & \text{by (2)} \\
 & = & (\Delta \setminus \boldsymbol{\varepsilon}_1*, (\boldsymbol{\varepsilon}_1*, x) \mapsto x)|_{\boldsymbol{\varepsilon}_2*} & \text{by def. of } \_, \_ \\
 & = & (\Delta \setminus \boldsymbol{\varepsilon}_1*)|_{\boldsymbol{\varepsilon}_2*}, (\boldsymbol{\varepsilon}_1* \mapsto x)|_{\boldsymbol{\varepsilon}_2*}, (x \mapsto x)|_{\boldsymbol{\varepsilon}_2*} & \text{by prop. of } \_|\_ \\
 & = & \Delta|_{\boldsymbol{\varepsilon}_2* \setminus \boldsymbol{\varepsilon}_1*}, (\boldsymbol{\varepsilon}_1* \mapsto x)|_{\boldsymbol{\varepsilon}_2*}, (x \mapsto x)|_{\boldsymbol{\varepsilon}_2*} & \text{by prop. of } \_|\_ \\
 & = & \Delta|_{\boldsymbol{\varepsilon}_2* \setminus \boldsymbol{\varepsilon}_1*}, (\boldsymbol{\varepsilon}_1* \cap \boldsymbol{\varepsilon}_2* \mapsto x), x \mapsto x & \text{by } x \in \boldsymbol{\varepsilon}_2, \text{ and prop. of } \_|\_ \\
 & = & \Delta|_{\boldsymbol{\varepsilon}_2* \setminus \boldsymbol{\varepsilon}_1*, x}, (\boldsymbol{\varepsilon}_1* \cap \boldsymbol{\varepsilon}_2* \mapsto x), x \mapsto x & \text{by } x \notin \mathrm{cod}\, \Delta
\end{array}$$

(ii) From $x \in \varepsilon_2$ and properties of reachability saturation, we have that $p* \subseteq \varepsilon_2*$.

(iii) From that, and $x \in \varepsilon_2$, and properties of saturation, we have either $p* \subseteq \varepsilon_1* \cap \varepsilon_2*$ or $p* \cap \varepsilon_1* \cap \varepsilon_2* = \varnothing$. Case distinction:

(iv) Case $p* \subseteq \varepsilon_1* \cap \varepsilon_2*$: Thus, $\varepsilon_1* \cap \varepsilon_2* = p*, q$ where $p* \cap q = \varnothing$

(A) That with (4).(b).(i) yields $\delta_2 = \Delta|_{\varepsilon_2*\setminus\varepsilon_1*,x}, (p*, q \mapsto x), x \mapsto x$

(B) Thus,

$$
\begin{aligned}
\delta_2[x \rightsquigarrow \Delta|_p] &= (\Delta|_{\varepsilon_2*\setminus\varepsilon_1*,x}, (p*, q \mapsto x), x \mapsto x)[x \rightsquigarrow \Delta|_{p*}] && \text{by (A)}\\
&= \Delta|_{\varepsilon_2*\setminus\varepsilon_1*,x}[x \rightsquigarrow \Delta|_{p*}], (p*, q \mapsto x)[x \rightsquigarrow \Delta|_{p*}], x \mapsto x[x \rightsquigarrow \Delta|_{p*}] && \text{by prop. of } \rightsquigarrow\\
&= \Delta|_{\varepsilon_2*\setminus\varepsilon_1*,x}[x \rightsquigarrow \Delta|_{p*}], (p*, q \mapsto x)[x \rightsquigarrow \Delta|_{p*}] && \text{by } x \notin \mathrm{dom}(\Delta|_{p*})\\
&= \Delta|_{\varepsilon_2*\setminus\varepsilon_1*,x}[x \rightsquigarrow \Delta|_{p*}], (p* \mapsto x)[x \rightsquigarrow \Delta|_{p*}] && \text{by (iv)}\\
&= \Delta|_{\varepsilon_2*\setminus\varepsilon_1*,x}[x \rightsquigarrow \Delta|_{p*}], \Delta|_{p*} && \text{by def. of } \rightsquigarrow\\
&= \Delta|_{\varepsilon_2*\setminus\varepsilon_1*,x}, \Delta|_{p*} && \text{by } x \notin \mathrm{cod}(\Delta)
\end{aligned}
$$

(C) From that, and (1), we conclude

$$
\begin{aligned}
\delta_1, \delta_2[x \rightsquigarrow \Delta|_{p*}] &= \Delta|_{\varepsilon_1*}, \Delta|_{\varepsilon_2*\setminus\varepsilon_1*,x}, \Delta|_{p*} &&\\
&= \Delta|_{\varepsilon_1*,\varepsilon_2*\setminus x,p*} && \text{by definition of \_|\_ and prop. of sets}\\
&= \Delta|_{\varepsilon_1*\triangleright\varepsilon_2*[p*/x]} && \text{by def. of substitution and effect composition}\\
&= \Delta|_{(\varepsilon_1\triangleright\varepsilon_2\theta)*} \quad \checkmark && \text{by prop. of saturation}
\end{aligned}
$$

(v) Case $p* \cap \varepsilon_1* \cap \varepsilon_2* = \varnothing$: Thus with (ii), we have $p* \subseteq \varepsilon_2* \setminus \varepsilon_1*$.

(A) Thus,

$$
\begin{aligned}
\delta_2[x \rightsquigarrow \Delta|_{p*}] &= (\Delta|_{\varepsilon_2*\setminus\varepsilon_1*,x}, (\varepsilon_1* \cap \varepsilon_2* \mapsto x), x \mapsto x)[x \rightsquigarrow \Delta|_{p*}] && \text{by (i)}\\
&= \Delta|_{\varepsilon_2*\setminus\varepsilon_1*,x}[x \rightsquigarrow \Delta|_{p*}] && \text{by (v) and } x \notin \mathrm{dom}(\Delta|_{p*})\\
&= \Delta|_{\varepsilon_2*\setminus\varepsilon_1*,x} && \text{by } x \notin \mathrm{cod}(\Delta)
\end{aligned}
$$

(B) From that, and (1), we conclude

$$
\begin{aligned}
\delta_1, \delta_2[x \rightsquigarrow \Delta|_{p*}] &= \Delta|_{\varepsilon_1*}, \Delta|_{\varepsilon_2*\setminus\varepsilon_1*,x} &&\\
&= \Delta|_{\varepsilon_1*,\varepsilon_2*\setminus x,p*} && \text{by (v)}\\
&= \Delta|_{\varepsilon_1*\triangleright\varepsilon_2*[p*/x]} && \text{by def. of substitution and effect composition}\\
&= \Delta|_{(\varepsilon_1\triangleright\varepsilon_2\theta)*} \quad \checkmark && \text{by prop. of saturation}
\end{aligned}
$$

□

The above proof of Lemma 5.1, justifies that we could alternatively pick the dependency $\Delta|_{(\varepsilon_1\triangleright\varepsilon_2\theta)*}$ as the synthesis result in the conclusion of rule ($\rightsquigarrow$-LET), and certifies that sequential dependency map composition and rewiring are consistent with effect composition and substitution, provided that the effects are saturated in the context.

Lemma 5.2 (Soundness of Synthesis). *Synthesis produces well-typed annotated programs:*

(1) If $[\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash n : T^{q}\ \varepsilon \rightsquigarrow \boldsymbol{n} \bullet \delta$, *then* $[\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash \boldsymbol{n} : T^{q}\ \varepsilon$.

(2) If $[\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash g : T^{q}\ \varepsilon \rightsquigarrow \boldsymbol{g} \bullet \delta$, *then* $[\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash \boldsymbol{g} : T^{q}\ \varepsilon$.

Proof. By straightforward mutual induction over the respective derivation, making use of Lemma 5.1 where appropriate. □

Lemma 5.3 (Synthesis is Total).

(1) If $[\Sigma \mid \Gamma]^{\varphi} \vdash_{\mathsf{M}} n : T^{q}\ \varepsilon$ *then* $\forall\Delta.\ \exists\boldsymbol{n}.\ \exists\delta.\ [\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash n : T^{q}\ \varepsilon \rightsquigarrow \boldsymbol{n} \bullet \delta$ *and* $\boldsymbol{n}$ *erases back to* n.

(2) If $[\Sigma \mid \Gamma]^{\varphi} \vdash_{\mathsf{M}} g : T^{q}\ \varepsilon$ *then* $\forall\Delta.\ \exists\boldsymbol{g}.\ \exists\delta.\ [\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash g : T^{q}\ \varepsilon \rightsquigarrow \boldsymbol{g} \bullet \delta$ *and* $\boldsymbol{g}$ *erases back to* g.

Proof. By straightforward mutual induction over the respective derivations, exploiting that typing rules of each system are in one-to-one correspondence. □

Corollary 5.4 (End-to-end Type Preservation). *For any well-formed context and compatible Δ, there is a type/effect/qualifier-preserving translation from the direct-style λ^*_ε-calculus into the λ^*_{G} graph IR.*

Proof. By lemmas 4.2, 5.3, and 5.2. □

5.4.2 Substitution and Rewiring.

Lemma 5.5 (Top-Level Rewiring and Substitution).
If $\Sigma \mid \Gamma, x : S^{p\cap r} \bullet \Delta.z$ ok, $[\Sigma \mid \varnothing]^{p} \bullet \varnothing \vdash \ell : S^{p}\ \boldsymbol{\varnothing}$, *and* $\theta = [p/x]$ *where* $p \subseteq dom(\Sigma)$ *and* $p \cap \varphi \subseteq p \cap r$, *and* $\mathrm{dom}(\delta) \subseteq \mathrm{dom}(\Sigma)$, *and* $\mathrm{cod}(\delta) \subseteq \{z\}$ *then*

$$\frac{[\Sigma \mid \Gamma, x : S^{p\cap r}]^{\varphi} \bullet \Delta \vdash n : T^{q}\ \boldsymbol{\varepsilon}}{[\Sigma \mid \Gamma\theta]^{\varphi\theta} \bullet \Delta[x \rightsquigarrow \delta \uparrow^{z}]\theta \vdash n[x \rightsquigarrow \delta]\theta[\ell/x]_{\mathrm{t}} : (T^{q}\ \boldsymbol{\varepsilon})\theta} \tag{1}$$

$$\frac{[\Sigma \mid \Gamma, x : S^{p\cap r}]^{\varphi} \bullet \Delta \vdash g : T^{q}\ \boldsymbol{\varepsilon}}{[\Sigma \mid \Gamma\theta]^{\varphi\theta} \bullet \Delta[x \rightsquigarrow \delta \uparrow^{z}]\theta \vdash g[x \rightsquigarrow \delta]\theta[\ell/x]_{\mathrm{t}} : (T^{q}\ \boldsymbol{\varepsilon})\theta} \tag{2}$$

Proof. The proof proceeds by mutual induction over the respective derivation. Ignoring dependencies, each case uses similar reasoning steps as the previous substitution lemma proof for the system with store-allocated values (Lemma 3.1). We focus here only on the interesting cases involving dependencies, which are the typing rules for λ-abstraction and let bindings.

- Case (n-abs): That is, $n = \lambda y.g \bullet \delta'$.
  1. We have $[\Sigma \mid \Gamma,\ x : S^{p\cap r},\ y : T^{p'}]^{q,y} \bullet \mapsto y \vdash g : U^{r'}\ \boldsymbol{\varepsilon}'$.
  2. We have $\delta' \sqsubseteq \boldsymbol{\varepsilon}'{*} \mapsto y$.
  3. By IH: $[\Sigma \mid \Gamma\theta,\ y : T\theta^{p'\theta}]^{q\theta,y} \bullet (\mapsto y[x \rightsquigarrow \delta \uparrow^{z}]\theta) \vdash g[x \rightsquigarrow \delta]\theta[\ell/x]_{\mathrm{t}} : (U^{r'}\ \boldsymbol{\varepsilon}')\theta$.
  4. Since $x \neq y$, it holds that $(\mapsto y[x \rightsquigarrow \delta \uparrow^{z}]\theta) = \mapsto y$.
  5. By (2), no entry in δ' points to x, because it is a submap of $\boldsymbol{\varepsilon}'{*} \mapsto y$ and $y \neq x$.
  6. Thus $\delta'[x \rightsquigarrow \delta]\theta = \delta'\theta$, and by (2) and monotonicity of substitution, $\delta'\theta \sqsubseteq \boldsymbol{\varepsilon}'\theta \mapsto y$.
  7. Hence $n[x \rightsquigarrow \delta]\theta[\ell/x]_{\mathrm{t}} = \lambda y.(g[x \rightsquigarrow \delta]\theta[\ell/x]_{\mathrm{t}}) \bullet \delta'\theta$.
  8. By (3),(6),(7), and (n-abs) the proof goal follows.
- Case (g-let): That is, $g = \mathbf{let}\ y = b \bullet \delta'\ \mathbf{in}\ g'$.
  1. We have $[\Sigma \mid \Gamma,\ x : S^{p\cap r}]^{\varphi} \bullet \Delta \vdash b : T^{q}\ \boldsymbol{\varepsilon}_1$.
  2. We have $[\Sigma \mid \Gamma,\ x : S^{p\cap r},\ y : T^{q}]^{\varphi,y} \bullet \Delta, (\boldsymbol{\varepsilon}_1{*}, y) \mapsto y \vdash g' : U^{r}\ \boldsymbol{\varepsilon}_2$.
  3. We have $\delta' \sqsubseteq \Delta|_{\boldsymbol{\varepsilon}_1{*}}$.
  4. By IH: $[\Sigma \mid \Gamma\theta]^{\varphi\theta} \bullet \Delta[x \rightsquigarrow \delta \uparrow^{z}]\theta \vdash b[x \rightsquigarrow \delta]\theta[\ell/x]_{\mathrm{t}} : (T^{q}\ \boldsymbol{\varepsilon}_1)\theta$.
  5. By IH: $[\Sigma \mid \Gamma\theta, y : T\theta^{q\theta}]^{\varphi\theta,y} \bullet (\Delta, (\boldsymbol{\varepsilon}_1{*}, y) \mapsto y)[x \rightsquigarrow \delta \uparrow^{z}]\theta \vdash g'[x \rightsquigarrow \delta]\theta[\ell/x]_{\mathrm{t}} : (U^{r}\ \boldsymbol{\varepsilon}_2)\theta$.
  6. From $x \neq y$ and the properties of rewiring and qualifier substitution on dependencies, it follows that
  $$(\Delta, (\boldsymbol{\varepsilon}_1{*}, y) \mapsto y)[x \rightsquigarrow \delta \uparrow^{z}]\theta = \Delta[x \rightsquigarrow \delta \uparrow^{z}]\theta, (\boldsymbol{\varepsilon}_1{*}\theta, y) \mapsto y.$$
  7. From (3) and monotonicity of substitution, we have $\delta'\theta \sqsubseteq \Delta\theta|_{\boldsymbol{\varepsilon}_1{*}\theta}$.
  8. By (4), (5), (6), (7), and (g-let) the proof goal follows.

□

5.4.3 Context Typing and Effect Introductions. We have generative effect introductions that modify the runtime context, and thus need lemmas for plugging/decomposition and growing the stack of dependencies by fresh effect dependencies from generative effects (i.e., reference allocations in this system).

Definition 5.6 (Context Typings). We define the typings of graph and binding contexts (Figure 10) relative to an ambient block start variable z, as follows:

(1) $[\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash z.G[\,\cdot\,] : S^{p}\ \varepsilon_1 \Rightarrow T^{q}\ \varepsilon_2$ iff $[\Sigma \mid \Gamma, x : S^{p}]^{\varphi} \bullet \Delta, x \mapsto z \vdash z.G[\,x\,] : T^{q}\ \varepsilon_2$ and $\varepsilon_1{*} = \mathrm{dom}(\delta)$ for the dependency δ at G's hole.

(2) $[\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash z.B[\,\cdot\,] : S^{p}\ \varepsilon_1 \Rightarrow T^{q}\ \varepsilon_2$ iff $[\Sigma \mid \Gamma, x : S^{p}]^{\varphi} \bullet \Delta, x \mapsto z \vdash z.B[\,x \bullet \delta\,] : T^{q}\ \varepsilon$ for some $\delta \sqsubseteq \Delta|_{\varepsilon_1 *}$.

Lemma 5.7 (Contextual Effect Propagation).

$$\frac{[\Sigma \mid \varnothing]^{\varphi} \bullet {\mapsto} z \vdash z.G[\,\cdot\,] : S^{p}\ \varepsilon_1 \Rightarrow T^{q}\ \varepsilon_2 \qquad \ell \notin \mathrm{dom}(\Sigma)}{[\Sigma, \ell : U^{r} \mid \varnothing]^{\varphi} \bullet {\mapsto} z \vdash z.G\langle \iota : z.\ell \rangle[\,\cdot\,] : S^{p}\ \varepsilon_1, \ell \Rightarrow T^{q}\ \varepsilon_2, \ell}$$

Proof. Let $[\Sigma \mid \varnothing]^{\varphi} \bullet {\mapsto} z \vdash z.G[\,\cdot\,] : S^{p}\ \varepsilon_1 \Rightarrow T^{q}\ \varepsilon_2$, and $\ell \notin \mathrm{dom}(\Sigma)$. We proceed by induction over G:

- Case $G = \square \bullet \delta$: Hence $G\langle \iota : z.\ell \rangle = \square \bullet \delta, \ell \mapsto z$, and $\Sigma \mid \Gamma \vdash S^{p}\ \varepsilon_1 <: T^{q}\ \varepsilon_2$, and $\varepsilon_1 = \mathrm{dom}(\delta)$. By the properties of subtyping, it holds that $\Sigma, \ell : U^{r} \mid \varnothing \vdash S^{p}\ \varepsilon_1, \ell <: T^{q}\ \varepsilon_2, \ell$. Using (G-RET) and (B-SUB) proves the goal.
- Case $G = (\mathbf{let}\ x = G'\ \mathbf{in}\ g) \bullet \delta$: Hence $G\langle \iota : z.\ell \rangle = (\mathbf{let}\ x = G'\langle \iota : z.\ell \rangle\ \mathbf{in}\ g) \bullet \delta, \ell \mapsto z$, and
  (1) By Def. 5.6 and typing inversion:
    (a) $[\Sigma \mid y : S^{p}]^{\varphi} \bullet {\mapsto} z \vdash z.G'[\,y\,] : S'^{p'}\ \varepsilon_2'$ for some $S'^{p'}\ \varepsilon_2'$.
    (b) $[\Sigma \mid y : S^{p}, x : S'^{p'}]^{\varphi} \bullet ({\mapsto} z, \varepsilon_2'{*} \mapsto x) \vdash g : T^{q'}\ \varepsilon_3$.
    (c) $q = q'[p'/x]$, $\varepsilon_2 = \varepsilon_2' \triangleright \varepsilon_3[p'/x]$.
    (d) $\varepsilon_1{*} = \mathrm{dom}(\delta')$ for the dependency δ' at the hole of G'.
  (2) By IH: $[\Sigma, \ell : U^{r} \mid \varnothing]^{\varphi} \bullet {\mapsto} z \vdash z.G'\langle \iota : z.\ell \rangle[\,\cdot\,] : S^{p}\ \varepsilon_1, \ell \Rightarrow S'^{p'}\ \varepsilon_2', \ell$.
  (3) By weakening on (1b): $[\Sigma, \ell : U^{r} \mid y : S^{p}, x : S'^{p'}]^{\varphi} \bullet ({\mapsto} z, (\varepsilon_2'{*}, \ell, r{*}) \mapsto x) \vdash g : T^{q'}\ \varepsilon_3$.
  (4) By (G-LET): $[\Sigma, \ell : U^{r} \mid y : S^{p}]^{\varphi} \bullet {\mapsto} z \vdash \mathbf{let}\ x = G'\langle \iota : z.\ell \rangle[\,y\,]\ \mathbf{in}\ g : T^{q'\theta}\ \varepsilon_2', \ell \triangleright \varepsilon_3\theta$ for $\theta = [p'/x]$, and the goal follows from that.

□

Lemma 5.8 (Decomposition).

(1) If $[\Sigma \mid \varnothing]^{\varphi} \bullet {\mapsto} z \vdash z.G[\,g\,] : T^{q}\ \varepsilon$, then $[\Sigma \mid \varnothing]^{\varphi} \bullet {\mapsto} z \vdash z.G[\,\cdot\,] : S^{p}\ \varepsilon' \Rightarrow T^{q}\ \varepsilon$ for some $S^{p}\ \varepsilon'$, and $[\Sigma \mid \varnothing]^{\varphi} \bullet {\mapsto} z \vdash z.g \bullet \delta : S^{p}\ \varepsilon'$, where δ is the dependency at the hole of G.

(2) If $[\Sigma \mid \varnothing]^{\varphi} \bullet {\mapsto} z \vdash z.B[\,b \bullet \delta\,] : T^{q}\ \varepsilon$, then $[\Sigma \mid \varnothing]^{\varphi} \bullet {\mapsto} z \vdash z.B[\,\cdot\,] : S^{p}\ \varepsilon' \Rightarrow T^{q}\ \varepsilon$ for some $S^{p}\ \varepsilon'$, where $[\Sigma \mid \varnothing]^{\varphi} \bullet {\mapsto} z \vdash b : S^{p}\ \varepsilon'$ and $\delta \sqsubseteq {\mapsto} z|_{\varepsilon' *}$.

Proof. Both cases are proved by induction over the respective context. □

Lemma 5.9 (Plugging).

(1) If $[\Sigma \mid \varnothing]^{\varphi} \bullet {\mapsto} z \vdash z.G[\,\cdot\,] : S^{p}\ \varepsilon_1 \Rightarrow T^{q}\ \varepsilon_2$ and $[\Sigma \mid \varnothing]^{\varphi} \bullet {\mapsto} z \vdash z.g \bullet \delta : S^{p}\ \varepsilon_1$ where δ is the dependency at the hole of G, then $[\Sigma \mid \varnothing]^{\varphi} \bullet {\mapsto} z \vdash z.G[\,g\,] : T^{q}\ \varepsilon_2$.

(2) If $[\Sigma \mid \varnothing]^{\varphi} \bullet {\mapsto} z \vdash z.B[\,\cdot\,] : S^{p}\ \varepsilon_1 \Rightarrow T^{q}\ \varepsilon_2$ and $[\Sigma \mid \varnothing]^{\varphi} \bullet {\mapsto} z \vdash b : S^{p}\ \varepsilon_1$, then for all $\delta \sqsubseteq \varepsilon_1{*} \mapsto z$ it holds that $[\Sigma \mid \varnothing]^{\varphi} \bullet {\mapsto} z \vdash z.B[\,b \bullet \delta\,] : T^{q}\ \varepsilon_2$.

Proof. Both cases are proved by induction over the respective context. □

Lemma 5.10 (Qualifier-Growing Replacement).

(1) If $[\Sigma \mid \varnothing]^{\varphi} \bullet \mapsto z \vdash z.G[\,g\,] : S^{p}\ \varepsilon_1 \Rightarrow T^{q}\ \varepsilon_2$ and $[\Sigma' \mid \varnothing]^{\varphi} \bullet \mapsto z \vdash g' : S^{p,r}\ \varepsilon_1, r$ where $\Sigma' \supseteq \Sigma$ and $r \subseteq \mathrm{dom}(\Sigma' \setminus \Sigma)$, then $[\Sigma' \mid \varnothing]^{\varphi} \bullet \mapsto z \vdash z.G'[\,g'\,] : T^{q,r}\ \varepsilon_2, r$ where G' is the result of contextual effect propagation for the fresh introduction form r.

(2) If $[\Sigma \mid \varnothing]^{\varphi} \bullet \mapsto z \vdash z.B[\,b \bullet \delta\,] : S^{p}\ \varepsilon_1 \Rightarrow T^{q}\ \varepsilon_2$ and $[\Sigma' \mid \varnothing]^{\varphi} \bullet \mapsto z \vdash b' : S^{p,r}\ \varepsilon_1, r$ where $\Sigma' \supseteq \Sigma$ and $r \subseteq \mathrm{dom}(\Sigma' \setminus \Sigma)$, then $[\Sigma' \mid \varnothing]^{\varphi} \bullet \mapsto z \vdash z.B'[\,b' \bullet \delta\,] : T^{q,r}\ \varepsilon_2, r$ where B' is the result of contextual effect propagation for the fresh introduction form r.

Proof. By decomposition (Lemma 5.8), contextual effect propagation (Lemma 5.7), and plugging (Lemma 5.9). □

5.4.4 Soundness.

Theorem 5.11 (Progress). *If $[\Sigma \mid \varnothing]^{\mathrm{dom}(\Sigma)} \vdash z.g \bullet \delta : T^{q}\ \varepsilon$, then either g is a location $\ell \in \mathrm{dom}(\Sigma)$, or for any store σ where $[\Sigma \mid \varnothing]^{\mathrm{dom}(\Sigma)} \vdash \sigma$, there exists a graph term g', store σ', and dependency δ' such that $\sigma \mid z.g \bullet \delta \longrightarrow_{\mathrm{G}} \sigma' \mid z.g' \bullet \delta'$.*

Proof. By induction over the typing derivation. □

Theorem 5.12 (Preservation).

$$\frac{[\Sigma \mid \varnothing]^{\mathrm{dom}(\Sigma)} \vdash z.g \bullet \delta : T^{q}\ \varepsilon \quad \Sigma \mid \varnothing \bullet \mapsto z.z\ \mathsf{ok} \quad [\Sigma \mid \varnothing]^{\mathrm{dom}(\Sigma)} \bullet \mapsto z \vdash \sigma \quad \sigma \mid z.g \bullet \delta \longrightarrow_{\mathrm{G}} \sigma' \mid z.g' \bullet \delta'}{\begin{array}{c}\exists \Sigma' \supseteq \Sigma.\ \exists p \subseteq \mathrm{dom}(\Sigma' \setminus \Sigma). \quad [\Sigma' \mid \varnothing]^{\mathrm{dom}(\Sigma')} \vdash z.g' \bullet \delta' : T^{q,p}\ \varepsilon, p \\ \Sigma' \mid \varnothing \bullet \mapsto z.z\ \mathsf{ok} \qquad [\Sigma' \mid \varnothing]^{\mathrm{dom}(\Sigma')} \bullet \mapsto z \vdash \sigma'\end{array}}$$

Proof. By inspecting the rule applied for the step $\sigma \mid z.g \bullet \delta \longrightarrow_{\mathrm{G}} \sigma' \mid z.g' \bullet \delta'$ and the qualifier-growing replacement Lemma 5.10, it is sufficient to prove that each rule is type/effect/qualifier preserving up to fresh store introductions in a minimal context:

- Case (β): In which case we have a well-typed application $\ell_1\ \ell_2 \bullet \delta_1$ in the hole. We proceed by induction over its typing derivation, which ends either in (n-app) or (b-sub). No store introduction occurs, hence the context type is preserved.
  - Case (n-app):
    1. We have $\mathrm{cod}(\delta_1) \subseteq \{z\}$.
    2. We have $\ell_1 : \left(x : T^{p* \cap q*} \to^{\varepsilon} U^{r}\right)^{q} \in [\Sigma \mid \varnothing]^{\mathrm{dom}(\Sigma)}$.
    3. We have $\ell_2 : T^{p} \in [\Sigma \mid \varnothing]^{\mathrm{dom}(\Sigma)}$.
    4. We have $x \notin \mathrm{fv}(U)$, $\varepsilon \subseteq q, x$, $r \subseteq \varphi, x$, and $\theta = [p/x]$.
    5. By (1) and environment relation, we have $\sigma(\ell_1) = \lambda x.g \bullet \delta_2$.
    6. By inversion: $[\Sigma \mid x : T'^{p'}]^{q',x} \bullet \mapsto x \vdash g : U'^{r'}\ \varepsilon'$, $\delta_2 \sqsubseteq \varepsilon' * \longmapsto x$, $\Sigma \mid \varnothing \vdash T^{p* \cap q*}\ \varnothing <: T'^{p'}\ \varnothing$, and $\Sigma \mid x : T^{p* \cap q*} \vdash U'^{r'}\ \varepsilon' <: U^{r}\ \varepsilon$, and $q' \subseteq q$.
    7. By narrowing, weakening, subsumption: $[\Sigma \mid x : T^{p* \cap q*}]^{\mathrm{dom}(\Sigma),x} \bullet \mapsto x \vdash g : U^{r}\ \varepsilon$.
    8. By (1) and the substitution and rewiring Lemma 5.5:
$$[\Sigma \mid \varnothing]^{\mathrm{dom}(\Sigma)} \bullet \mapsto z \vdash g[x \rightsquigarrow \delta_1][\ell_2/x][\ell_2/x]_{\mathrm{t}} : U^{r[p/x]}\ \varepsilon[p/x].$$
    9. By (6) and transitivity, we have $\delta_2 \sqsubseteq \varepsilon * \longmapsto x$, and hence
$$\delta_1[x \rightsquigarrow \delta_1][\ell_2/x][\ell_2/x]_{\mathrm{t}} \sqsubseteq \varepsilon * [p/x] \longmapsto z$$
    10. By (8), (9), and plugging Lemma 5.9 we can now prove this case.
  - Case (b-sub): By IH and subsumption.
- Case (let): Follows from the substitution and rewiring Lemma 5.5.
- Case (intro): In which case we have a well-typed let binding **let** $x = \iota \bullet \delta$ **in** g in the hole. By induction over the derivation, only (b-sub) or (g-let) applies. The first is trivial, and we consider the latter case. By the contextual effect propagation Lemma 5.7, we have increased the qualifiers and effects of the right-hand-side context typing with the fresh new store

location, after which we can proceed as in the (LET) case with the substitution and rewiring Lemma 5.5 and conclude.

- Case (DEREF): In which case we have a well-typed dereference $!\,\ell \bullet \delta$ with $\mathrm{cod}(\delta) \subseteq \{z\}$ in the hole. By induction over the derivation, only (B-SUB) or (N-!) applies. The first is again trivial, and the latter case is straightforward, since by the environment predicate we have that thew hole is plugged with a store value of the same type.
- Case (ASSIGN): Similar to the previous case.

□

Corollary 5.13 (Preservation of Separation). *Interleaved executions preserve types and disjointness:*

$$\frac{\begin{array}{lllll} [\Sigma \mid \varnothing]^{\mathrm{dom}(\Sigma)} \vdash z.g_1 \bullet \delta_1 : T_1^{q_1}\ \varepsilon_1 & \sigma\ \mid z.g_1 \bullet \delta_1 \longrightarrow_{\mathrm{G}} \sigma' \ \mid z.g_1' \bullet \delta_1{}' & [\Sigma \mid \varnothing]^{\mathrm{dom}(\Sigma)} \vdash \sigma & \Sigma \mid \varnothing \bullet \mapsto z.z \text{ ok} \\ [\Sigma \mid \varnothing]^{\mathrm{dom}(\Sigma)} \vdash z.g_2 \bullet \delta_2 : T_2^{q_2}\ \varepsilon_2 & \sigma' \mid z.g_2 \bullet \delta_2 \longrightarrow_{\mathrm{G}} \sigma'' \mid z.g_2' \bullet \delta_2{}' & q_1 \cap q_2 \subseteq \varnothing \end{array}}{\begin{array}{ll} \exists p_1\ p_2\ \varepsilon_1'\ \varepsilon_2'\ \Sigma'\ \Sigma''. & [\Sigma' \ \mid \varnothing]^{\mathrm{dom}(\Sigma')} \vdash z.g_1' \bullet \delta_1{}' : T_1^{p_1}\ \varepsilon_1' \quad \Sigma'' \supseteq \Sigma' \supseteq \Sigma \\ & [\Sigma'' \mid \varnothing]^{\mathrm{dom}(\Sigma'')} \vdash z.g_2' \bullet \delta_2{}' : T_2^{p_2}\ \varepsilon_2' \quad p_1 \cap p_2 \subseteq \varnothing \end{array}}$$

Corollary 5.14 (Dependency Safety). *Evaluation respects the order of effect dependencies for well-typed graph IR terms,* i.e., *an effectful graph node is executed only if all its dependencies are resolved in the store.*

6 EXTENSION WITH SOFT DEPENDENCIES

We extend the graph IR λ^*_{G} from the previous section with soft dependencies. During code generation, a node that is only soft-depended by other nodes is considered dead, and therefore is not scheduled (cf. Section 9). If node A hard-depends on node B, then B must be executed (or scheduled) before A. This is the default notion of dependency for the base λ^*_{G} system (Section 5), and entails that no effect operation can be skipped. This is evidently too rigid, and as motivated in the main paper, soft dependencies gives us more slack to outright omit effects that are not observable, *e.g.*, write-after-write (WAR) on a mutable reference cell. If A *soft-depends* on B, then B should never be scheduled after A, but B might not be scheduled even if A is scheduled. Being able to tell that some effectful part of a higher-order program can be omitted is immensely useful.

The entire formal system and reasoning principles of λ^*_{G} carry over into a system with hard and soft dependencies as presented in this section. The difference is the change in the effect and dependency structure, *i.e.*, effects are split into reads and writes, which induce hard dependencies (the previous section's notion) and soft dependencies, respectively. That is to say, we can regard these new structures as a product composition of the previous with new structures.

In future work, we would like to develop a generic theory of graph IRs that is parametric in such effect and dependency structures. Bao et al. [2021]'s direct style system already proposes one half of the solution by adopting Gordon [2021]'s effect quantales. We anticipate that a general graph IR would require a "dependency quantuale" that mirrors a given effect quantale.

In the following, we focus on the key differences to the previous section.

6.1 Effects and Dependencies for Reads and Writes

The nature of effects changes from a simple "effectful use of/on a variable" to a more refined distinction, classifying the effect on the variables as either a read or a write effect. Due to aliasing/reachability, there is usually more than one variable involved, and compound expressions accumulate their effects. Thus, we change the effect domain to labelled pairs $\mathsf{r} : q; \mathsf{w} : p$ of qualifiers, grouping variables/locations by read and write (Figure 11). We also lift the preexisting operations and relations involving effects to such pairs in a straightforward manner, with the intent that the

Graph IR $\boxed{\lambda^*_{\mathsf{G}}}$

H, h	$::=$	$\overline{\mathsf{x} \mapsto \mathsf{x}}$	Hard Dependencies
S, s	$::=$	$\mathsf{x} \mapsto \overline{\mathsf{x}}$	Soft Dependencies
Δ, δ	$::=$	$\mathsf{h}; \mathsf{s}$	Dependencies
$\boldsymbol{\varepsilon}$	$::=$	$\mathsf{r}{:}q; \mathsf{w}{:}q$	Effects

Effects and Dependencies

$(\mathsf{r} : q_1 \,;\, \mathsf{w} : p_1) \rhd (\mathsf{r} : q_2; \mathsf{w} : p_2)$	$:= (\mathsf{r} : q_1, q_2 \,;\, \mathsf{w} : p_1, p_2)$	Sequential Composition
$(\mathsf{r} : q \,;\, \mathsf{w} : p) \subseteq r$	$:= q, p \subseteq r$	Effect/Qualifier Inclusion
$\Sigma \mid \Gamma \vdash (\mathsf{r} : q \,;\, \mathsf{w} : p){*}$	$:= \Sigma \mid \Gamma \vdash (\mathsf{r} : q{*} \,;\, \mathsf{w} : p{*})$	Effect Saturation
$\boldsymbol{\varnothing}$	$:= (\mathsf{r} : \varnothing \,;\, \mathsf{w} : \varnothing)$	Purity
$\mathsf{r}(q)$	$:= (\mathsf{r} : q \,;\, \mathsf{w} : \varnothing)$	Just a Read
$\mathsf{w}(q)$	$:= (\mathsf{r} : \varnothing \,;\, \mathsf{w} : q)$	Just a Write
$(\mathsf{h}_1; \mathsf{s}_1), (\mathsf{h}_2; \mathsf{s}_2)$	$:= (\mathsf{h}_1, \mathsf{h}_2); (\mathsf{s}_1, \mathsf{s}_2)$	Update
$(\mathsf{h}; \mathsf{s})\vert_{(\mathsf{r}:q\,;\,\mathsf{w}:p)}$	$:= (\mathsf{h}\vert_q; \mathsf{h}\vert_p \sqcup \mathsf{s}\vert_p)$	Restriction
$(\mathsf{h}_1; \mathsf{s}_1)[x \rightsquigarrow \mathsf{h}_2; \mathsf{s}_2]$	$:= (\mathsf{h}_1[x \rightsquigarrow \mathsf{h}_2]; \mathsf{s}_1[x \rightsquigarrow \mathsf{s}_2])$	Rewiring
$\mathsf{s} \oplus_x q$	$:= \mathsf{s}, \{y \mapsto \mathsf{s}(y), x \mid y \in q\}$	Insertion of x
$(\mathsf{h}; \mathsf{s}) \oplus_x (\mathsf{r} : q \,;\, \mathsf{w} : p)$	$:= (\mathsf{h}, (p, x) \mapsto x \,;\, (\mathsf{s}, (p, x) \mapsto \varnothing) \oplus_x q$	Last Use at x

Dependency Synthesis $\boxed{[\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash (n \mid g) : T^{q}\, \boldsymbol{\varepsilon} \rightsquigarrow (\boldsymbol{n} \mid \boldsymbol{g}) \bullet \delta}$

$$\frac{\begin{array}{c}\mathsf{x} : \mathsf{Alloc}^{q} \in [\Sigma \mid \Gamma]^{\varphi} \\ \mathsf{y} : B^{\varnothing} \in [\Sigma \mid \Gamma]^{\varphi}\end{array}}{\begin{array}{c}[\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash \mathbf{ref}_{\mathsf{x}}\ \mathsf{y} : (\mathsf{Ref}\ B)^{\varnothing}\ \mathsf{r}(\mathsf{x}) \\ \rightsquigarrow \mathbf{ref}_{\mathsf{x}}\ \mathsf{y} \bullet \Delta\vert_{\mathsf{r}(\mathsf{x}){*}}\end{array}}\ (\rightsquigarrow\text{-REF})$$

$$\frac{\begin{array}{c}\mathsf{x} : (\mathsf{Ref}\ B)^{q} \in [\Sigma \mid \Gamma]^{\varphi} \\ \mathsf{y} : B^{\varnothing} \in [\Sigma \mid \Gamma]^{\varphi}\end{array}}{\begin{array}{c}[\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash \mathsf{x} := \mathsf{y} : \mathsf{Unit}^{\varnothing}\ \mathsf{w}(\mathsf{x}) \\ \rightsquigarrow \mathsf{x} := \mathsf{y} \bullet \Delta\vert_{\mathsf{w}(\mathsf{x}){*}}\end{array}}\ (\rightsquigarrow\text{-:=})$$

$$\frac{\mathsf{x} : (\mathsf{Ref}\ B)^{q} \in [\Sigma \mid \Gamma]^{\varphi}}{\begin{array}{c}[\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash\ !\,\mathsf{x} : B^{\varnothing}\ \mathsf{r}(\mathsf{x}) \\ \rightsquigarrow\ !\,\mathsf{x} \bullet \Delta\vert_{\mathsf{r}(\mathsf{x}){*}}\end{array}}\ (\rightsquigarrow\text{-!})$$

$$\frac{\begin{array}{c}[\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash b : S^{p}\ \boldsymbol{\varepsilon_1} \rightsquigarrow \boldsymbol{b} \bullet \delta_1 \\ [\Sigma \mid \Gamma, x : S^{p}]^{\varphi, x} \bullet \Delta \oplus_x \boldsymbol{\varepsilon_1}{*} \vdash g : T^{q}\ \boldsymbol{\varepsilon_2} \\ \rightsquigarrow \boldsymbol{g} \bullet \delta_2 \\ \theta = [p/x] \quad x \notin \mathrm{fv}\,(T)\end{array}}{\begin{array}{c}[\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash \mathbf{let}\ x = b\ \mathbf{in}\ g : (T^{q}\ \boldsymbol{\varepsilon_1} \rhd \boldsymbol{\varepsilon_2})\theta \\ \rightsquigarrow (\mathbf{let}\ x = \boldsymbol{b} \bullet \delta_1\ \mathbf{in}\ \boldsymbol{g}) \bullet \delta_1, \delta_2[x \rightsquigarrow \Delta\vert_{p*}]\end{array}}\ (\rightsquigarrow\text{-LET})$$

Effect Subtyping $\boxed{\Sigma \mid \Gamma \vdash \boldsymbol{\varepsilon_1} <: \boldsymbol{\varepsilon_2}}$

$$\frac{\Sigma \mid \Gamma \vdash q_1 <: q_2 \qquad \Sigma \mid \Gamma \vdash p_1 <: p_2}{\Sigma \mid \Gamma \vdash (\mathsf{r} : q_1 \,;\, \mathsf{w} : p_1) <: (\mathsf{r} : q_2; \mathsf{w} : p_2)}\ (\text{E-SUB})$$

Fig. 11. The syntax and typing rules of the graph IR λ^*_{G} with hard and soft dependencies. We only show the changes relative to Figure 9. All other rules remain exactly the same using overloaded operations on effects and qualifiers that act component-wise on hard and soft dependencies.

λ^*_{G} typing and synthesis rules can be copied over almost one-to-one with overloaded notations for the new effect structure. The only tweak needed is that typing rules introducing effects should classify them as read or write effects, and the last-use update at let bindings is a bit more involved.

6.2 Hard-and-Soft Dependency Calculation

We use the hard dependency in Δ to track the last write of any variable/location in context, whereas its soft dependency tracks all the reads on a variable since it was last written.

Projections need to merge the hard dependencies of a write effect into its soft dependencies, and project the hard dependencies for reads, *i.e.*, $(\mathsf{h}; \mathsf{s})|_{(\mathsf{r}:q\,;\,\mathsf{w}:p)} := (\mathsf{h}|_q; \mathsf{h}|_p \sqcup \mathsf{s}|_p)$.

Last-use updates at let bindings need to reset the recorded last reads for any written variable, and add the bound variable to the last reads for any written variable, *i.e.*,

$$(\mathsf{h}; \mathsf{s}) \oplus_x (\mathsf{r} : q\,;\, \mathsf{w} : p) := (\mathsf{h}, (p, x) \mapsto x\,;\ (\mathsf{s}, (p, x) \mapsto \varnothing) \oplus_x q),$$

where x is the let-bound variable and $\mathsf{s} \oplus_x q := \mathsf{s}, \{y \mapsto \mathsf{s}(y), x \mid y \in q\}$ adds x into each set pointed to by q.

6.3 Statics

Figure 11 shows the required changes to the synthesis rules from Figure 9 (and also indicates the needed changes for the checking rules from Figure 8). With the overloaded operations on effects and dependencies, the rules for mutable references need to classify their effects on the operands. That is, reference allocations cause a read on the used allocation capability and its reachable aliases, dereferences a read on the target reference and its aliases, and assignments a write, accordingly.

6.4 Metatheory

The theorems and proofs for the graph IR with hard and soft dependency are for the most part identical to the previous system, and we just repeat the most relevant theorems without proof.

Lemma 6.1 (Synthesis Invariant). *Dependencies are completely determined by the context and effect, as follows:*

(1) If $[\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash n : T^{q}\ \varepsilon \rightsquigarrow \boldsymbol{n} \bullet \delta$, *then* $\delta = \Delta|_{\varepsilon*}$.
(2) If $[\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash g : T^{q}\ \varepsilon \rightsquigarrow \boldsymbol{g} \bullet \delta$, *then* $\delta = \Delta|_{\varepsilon*}$.

Proof. Analogous to the proof of Lemma 5.1. □

Lemma 6.2 (Soundness of Synthesis). *Synthesis produces well-typed annotated programs:*

(1) If $[\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash n : T^{q}\ \varepsilon \rightsquigarrow \boldsymbol{n} \bullet \delta$, *then* $[\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash \boldsymbol{n} : T^{q}\ \varepsilon$.
(2) If $[\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash g : T^{q}\ \varepsilon \rightsquigarrow \boldsymbol{g} \bullet \delta$, *then* $[\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash \boldsymbol{g} : T^{q}\ \varepsilon$.

Proof. Analogous to the proof of Lemma 5.2. □

Lemma 6.3 (Synthesis is Total).

(1) If $[\Sigma \mid \Gamma]^{\varphi} \vdash_{\mathsf{M}} n : T^{q}\ \varepsilon$ *then* $\forall \Delta.\ \exists \boldsymbol{n}.\ \exists \delta.\ [\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash n : T^{q}\ \varepsilon \rightsquigarrow \boldsymbol{n} \bullet \delta$ *and* $\boldsymbol{n}$ *erases back to* n.
(2) If $[\Sigma \mid \Gamma]^{\varphi} \vdash_{\mathsf{M}} g : T^{q}\ \varepsilon$ *then* $\forall \Delta.\ \exists \boldsymbol{g}.\ \exists \delta.\ [\Sigma \mid \Gamma]^{\varphi} \bullet \Delta \vdash g : T^{q}\ \varepsilon \rightsquigarrow \boldsymbol{g} \bullet \delta$ *and* $\boldsymbol{g}$ *erases back to* g.

Proof. Analogous to the proof of Lemma 5.3. □

Corollary 6.4 (End-to-end Type Preservation). *For any well-formed context and compatible* Δ, *there is a type/effect/qualifier-preserving translation from the direct-style* λ^*_{ε}*-calculus into the* λ^*_{G} *graph IR with hard and soft dependencies.*

Proof. By lemmas 4.2, 6.3, and 6.2. □

Theorem 6.5 (Progress). *If $[\Sigma \mid \varnothing]^{\mathrm{dom}(\Sigma)} \vdash z.g \bullet \delta : T^{q}\ \varepsilon$, then either g is a location $\ell \in \mathrm{dom}(\Sigma)$, or for any store σ where $[\Sigma \mid \varnothing]^{\mathrm{dom}(\Sigma)} \vdash \sigma$, there exists a graph term g', store σ', and dependency δ' such that $\sigma \mid z.g \bullet \delta \longrightarrow_{\mathrm{G}} \sigma' \mid z.g' \bullet \delta'$.*

Theorem 6.6 (Preservation).

$$\frac{[\Sigma \mid \varnothing]^{\mathrm{dom}(\Sigma)} \vdash z.g \bullet \delta : T^{q}\ \varepsilon \quad \Sigma \mid \varnothing \bullet \mapsto z.z\ \mathrm{ok} \quad [\Sigma \mid \varnothing]^{\mathrm{dom}(\Sigma)} \bullet \mapsto z \vdash \sigma \quad \sigma \mid z.g \bullet \delta \longrightarrow_{\mathrm{G}} \sigma' \mid z.g' \bullet \delta'}{\begin{array}{c} \exists \Sigma' \supseteq \Sigma.\ \exists p \subseteq \mathrm{dom}(\Sigma' \setminus \Sigma).\quad [\Sigma' \mid \varnothing]^{\mathrm{dom}(\Sigma')} \vdash z.g' \bullet \delta' : T^{q,p}\ \varepsilon, p \\ \Sigma' \mid \varnothing \bullet \mapsto z.z\ \mathrm{ok} \qquad [\Sigma' \mid \varnothing]^{\mathrm{dom}(\Sigma')} \bullet \mapsto z \vdash \sigma' \end{array}}$$

Corollary 6.7 (Preservation of Separation). *Interleaved executions preserve types and disjointness:*

$$\frac{\begin{array}{l} [\Sigma \mid \varnothing]^{\mathrm{dom}(\Sigma)} \vdash g_1 \bullet \delta_1 : T_1^{q_1}\ \varepsilon_1 \quad \sigma\ \mid z.g_1 \bullet \delta_1 \longrightarrow_{\mathrm{G}} \sigma'\ \mid z.g_1' \bullet \delta_1{}' \quad [\Sigma \mid \varnothing]^{\mathrm{dom}(\Sigma)} \vdash \sigma \quad \Sigma \mid \varnothing \bullet \mapsto z.z\ \mathrm{ok} \\ [\Sigma \mid \varnothing]^{\mathrm{dom}(\Sigma)} \vdash g_2 \bullet \delta_2 : T_2^{q_2}\ \varepsilon_2 \quad \sigma' \mid z.g_2 \bullet \delta_2 \longrightarrow_{\mathrm{G}} \sigma'' \mid z.g_2' \bullet \delta_2{}' \quad q_1 \cap q_2 \subseteq \varnothing \end{array}}{\begin{array}{l} \exists p_1\ p_2\ \varepsilon_1'\ \varepsilon_2'\ \Sigma'\ \Sigma''.\quad [\Sigma'\ \mid \varnothing]^{\mathrm{dom}(\Sigma')} \vdash z.g_1' \bullet \delta_1{}' : T_1^{p_1}\ \varepsilon_1' \quad \Sigma'' \supseteq \Sigma' \supseteq \Sigma \\ \qquad\qquad\qquad\qquad [\Sigma'' \mid \varnothing]^{\mathrm{dom}(\Sigma'')} \vdash z.g_2' \bullet \delta_2{}' : T_2^{p_2}\ \varepsilon_2' \quad p_1 \cap p_2 \subseteq \varnothing \end{array}}$$

Corollary 6.8 (Dependency Safety). *Evaluation respects the order of effect dependencies for well-typed graph IR terms,* i.e., *an effectful graph node is executed only if all its dependencies are resolved in the store.*

7 CONTEXTUAL EQUIVALENCE - THE DIRECT-STYLE $\lambda^{*}_{\varepsilon}$-CALCULUS

We apply a logical relations approach following [Ahmed et al. 2009; Benton et al. 2007; Timany et al. 2022] to support relational reasoning with respect to the *observational equivalence* of two programs. We define binary logical relations over reachability types (the $\lambda^{*}_{\varepsilon}$-calculus in Sec. 2), and prove the soundness of the equational rules. Our development is based on a framework for modeling reachability types with logical relations developed in parallel with this work [Bao et al. 2023]. To make the present report self-contained, pieces of Bao et al. [2023] are repeated in this section without further reference. To avoid technical complications, we choose a model that allows mutable references to contain only first-order values, consistent with the previous sections. The definition of the logical relation can be extended to support higher-order references using well-established techniques such as step-indexing [Ahmed et al. 2009; Ahmed 2004; Appel and McAllester 2001], which we leave as future work.

7.1 High-level Overview of the Proofs

A program t_1 is said to be *contextually equivalent* to another program t_2, written as $\Gamma^{\varphi} \models t_1 \approx_{\mathrm{ctx}} t_2 : T^{p}\ \varepsilon$, if for any program context C with a hole of type $T^{p}\ \varepsilon$, if $C[t_1]$ has some (observable) behavior, then so does $C[t_2]$. The definition of context C can be found in Sec. 7.2.

Following the approach of Timany et al. [2022] and related prior works [Ahmed et al. 2009], we define a judgement for logical equivalence using binary logical relations, written as $\Gamma^{\varphi} \models t_1 \approx_{\log} t_2 : T^{q}\ \varepsilon$.

The high-level structure of the proof is the following:

- Soundness (Theorem 7.43, Sec. 7.8). We show that the logical relation is sound with respect to contextual equivalence:
$$\Gamma^{\varphi} \models t_1 \approx_{\log} t_2 : T^{q}\ \varepsilon \text{ implies } \Gamma^{\varphi} \models t_1 \approx_{\mathrm{ctx}} t_2 : T^{q}\ \varepsilon.$$
- Compatibility lemmas (Sec. 7.7). We show that the logical relation is compatible with syntactic typing.

These results can be used to prove the soundness of the equational rules (Sec. 7.9).

Context for Contextual Equivalence

$$C \;::=\; \square \mid C\,t \mid t\,C \mid \lambda x.C \mid \mathbf{ref}_t\; C \mid \mathbf{ref}_C\; t \mid\; !\,C \mid C := t \mid t := C \mid \mathbf{let}\; x = C\; \mathbf{in}\; t \mid \mathbf{let}\; x = t\; \mathbf{in}\; C$$

Context Typing Rules $\boxed{C : (\Gamma^{\varphi}; T^{q}\ \varepsilon) \Rrightarrow (\Gamma'^{\varphi}; T^{q}\ \varepsilon)}$

$$\dfrac{\Gamma^{\varphi} \vdash T^{q}\ \varepsilon <: T'^{q'}\ \varepsilon'}{\square : (\Gamma^{\varphi}; T^{q}\ \varepsilon) \Rrightarrow (\Gamma'^{\varphi'}; T'^{q'}\ \varepsilon')} \quad (\textsc{c-hole})$$

$$\dfrac{\begin{array}{c} C : (\Gamma^{\varphi}; U^{r}\ \varepsilon_1) \Rrightarrow (\Gamma'^{\varphi'}; ((x : T^{p*\cap q*}) \to^{\varepsilon_3} U'^{r'})^{q'}\ \varepsilon_4) \quad \Gamma'^{\varphi'} \vdash t_2 : T^{p}\ \varepsilon_2 \\ x \notin \mathrm{fv}(U') \quad r' \subseteq \varphi', x \quad \varepsilon_3 \subseteq \varphi', x \quad \theta = [p/x] \end{array}}{C\ t_2 : (\Gamma^{\varphi}; U^{r}\ \varepsilon_1) \Rrightarrow (\Gamma'^{\varphi'}; (U'^{r'}\ \varepsilon_4 \rhd \varepsilon_2 \rhd \varepsilon_3)\theta)} \quad (\textsc{c-app-1})$$

$$\dfrac{\begin{array}{c} \Gamma'^{\varphi'} \vdash t_1 : ((x : T^{p*\cap q*}) \to^{\varepsilon_4} U'^{r'})^{q'}\ \varepsilon_2 \quad C : (\Gamma^{\varphi}; U^{r}\ \varepsilon_1) \Rrightarrow (\Gamma'^{\varphi'}; T^{p}\ \varepsilon_3) \\ x \notin \mathrm{fv}(U') \quad r' \subseteq \varphi', x \quad \varepsilon_3 \subseteq \varphi', x \quad \theta = [p/x] \end{array}}{t_1\ C : (\Gamma^{\varphi}; U^{r}\ \varepsilon_1) \Rrightarrow (\Gamma'^{\varphi'}; (U'^{r'}\ \varepsilon_2 \rhd \varepsilon_3 \rhd \varepsilon_4)\theta)} \quad (\textsc{c-app-2})$$

$$\dfrac{C : (\Gamma^{\varphi}; S^{r}\ \varepsilon) \Rrightarrow ((\Gamma'\ ,\ x : T^{p})^{q,x}; U^{r}\ \varepsilon') \quad q \subseteq \varphi}{\lambda x.C : (\Gamma^{\varphi}; S^{r}\ \varepsilon) \Rrightarrow (\Gamma'^{\varphi'}; ((x : T^{p}) \to^{\varepsilon'} U^{r})^{q}\ \varnothing)} \quad (\textsc{c-}\lambda)$$

$$\dfrac{\Gamma'^{\varphi'} \vdash t : \mathsf{Alloc}^{q}\ \varepsilon_1 \quad C : (\Gamma^{\varphi}; T^{r}\ \varepsilon_2) \Rrightarrow (\Gamma'^{\varphi'}; B^{\varnothing}\ \varepsilon')}{\mathbf{ref}_t\ C : (\Gamma^{\varphi}; T^{r}\ \varepsilon_2) \Rrightarrow (\Gamma'^{\varphi'}; \mathsf{Ref}\ B^{\varnothing}\ \varepsilon_1 \rhd \varepsilon' \rhd q)} \quad (\textsc{c-ref-1})$$

$$\dfrac{\Gamma'^{\varphi'} \vdash t : B^{\varnothing}\ \varepsilon_2 \quad C : (\Gamma^{\varphi}; T^{r}\ \varepsilon_1) \Rrightarrow (\Gamma'^{\varphi'}; \mathsf{Alloc}^{q}\ \varepsilon_1')}{\mathbf{ref}_C\ t : (\Gamma^{\varphi}; T^{r}\ \varepsilon_1) \Rrightarrow (\Gamma'^{\varphi'}; \mathsf{Ref}\ B^{\varnothing}\ \varepsilon_1' \rhd \varepsilon_2 \rhd q)} \quad (\textsc{c-ref-2})$$

$$\dfrac{C : (\Gamma^{\varphi}; T^{r}\ \varepsilon) \Rrightarrow (\Gamma'^{\varphi'}; (\mathsf{Ref}\ B)^{q'}\ \varepsilon')}{\vdash !\,C : (\Gamma^{\varphi}; T^{r}\ \varepsilon) \Rrightarrow (\Gamma'^{\varphi'}; B^{\varnothing}\ \varepsilon')} \quad (\textsc{c-!})$$

$$\dfrac{C : (\Gamma^{\varphi}; T^{r}\ \varepsilon) \Rrightarrow (\Gamma'^{\varphi'}; (\mathsf{Ref}\ B)^{q'}\ \varepsilon') \quad \Gamma'^{\varphi'} \vdash t_2 : B^{\varnothing}\ \varepsilon'}{C := t_2 : (\Gamma^{\varphi}; T^{r}\ \varepsilon) \Rrightarrow (\Gamma'^{\varphi'}; \mathsf{Unit}^{\varnothing}\ \varepsilon')} \quad (\textsc{c-:=-1})$$

$$\dfrac{\Gamma'^{\varphi'} \vdash t_1 : (\mathsf{Ref}\ B)^{q'}\ \varepsilon' \quad C : (\Gamma^{\varphi}; T^{r}\ \varepsilon) \Rrightarrow (\Gamma'^{\varphi'}; B^{\varnothing}\ \varepsilon')}{t_1 := C : (\Gamma^{\varphi}; T^{r}\ \varepsilon) \Rrightarrow (\Gamma'^{\varphi'}; \mathsf{Unit}^{\varnothing}\ \varepsilon')} \quad (\textsc{c-:=-2})$$

$$\dfrac{\begin{array}{c} C : (\Gamma^{\varphi}; U^{r}\ \varepsilon_1) \Rrightarrow ((\Gamma', x : T^{p*\cap\varphi'*})^{\varphi',x}; U'^{r'}\ \varepsilon_2) \quad \Gamma'^{\varphi'} \vdash t : T^{p}\ \varepsilon_3 \\ x \notin \mathrm{fv}(U') \quad \theta = [p/x] \end{array}}{\mathbf{let}\ x = t\ \mathbf{in}\ C : (\Gamma^{\varphi}; U^{r}\ \varepsilon_1) \Rrightarrow (\Gamma'^{\varphi'}; (U'^{r'}\ \varepsilon_2 \rhd \varepsilon_3)\theta)} \quad (\textsc{c-let-1})$$

$$\dfrac{\begin{array}{c} (\Gamma', (x : T^{p*\cap\varphi'*}))^{\varphi',x} \vdash t : U'^{r'}\ \varepsilon_2 \quad C : (\Gamma^{\varphi}; U^{r}\ \varepsilon_1) \Rrightarrow (\Gamma'^{\varphi'}; T^{p}\ \varepsilon_3) \\ x \notin \mathrm{fv}(U') \quad \theta = [p/x] \end{array}}{\mathbf{let}\ x = C\ \mathbf{in}\ t : (\Gamma^{\varphi}; U^{r}\ \varepsilon_1) \Rrightarrow (\Gamma'^{\varphi'}; (U'^{r'}\ \varepsilon_2 \rhd \varepsilon_3)\theta)} \quad (\textsc{c-let-2})$$

Fig. 12. Context typing rules for the λ^*_ε-Calculus.

7.2 Contextual Equivalence

Unlike reduction contexts E in Fig. 4, contexts C for reasoning about the equivalence allow a "hole" to appear in any place. We write $C : (\Gamma^{\varphi}; T^{q}\ \varepsilon) \Rrightarrow (\Gamma'^{\varphi'}; T'^{q'}\ \varepsilon')$ to mean that the context C is a program of type $T'^{q'}\ \varepsilon'$ (closed under $\Gamma'^{\varphi'}$) with a hole that can be filled with any program of type

$T^{q}\ \boldsymbol{\varepsilon}$ (closed under Γ^{φ}). The typing rules for well-typed contexts imply that if $\Gamma^{\varphi} \vdash t : T^{q}\ \boldsymbol{\varepsilon}$ and $C : (\Gamma^{\varphi}; T^{q}\ \boldsymbol{\varepsilon}) \Rrightarrow (\Gamma'^{\varphi'}; T'^{q'}\ \boldsymbol{\varepsilon'})$ hold, then $\Gamma'^{\varphi'} \vdash C[t] : T'^{q'}\ \boldsymbol{\varepsilon'}$. Fig. 12 shows the typing rules for well-typed contexts.

Two well-typed terms, t_1 and t_2, under type context Γ^{φ}, are *contextually equivalent* if any occurrences of the first term in a closed term can be replaced by the second term without affecting the *observable results* of reducing the program, which is formally defined as follows:

Definition 7.1 (Contextual Equivalence). We say t_1 is *contextually equivalent* to t_2, written as $\Gamma^{\varphi} \models t_1 \approx_{\text{ctx}} t_2 : T^{p}\ \boldsymbol{\varepsilon}$, if $\Gamma^{\varphi} \vdash t_1 : T^{q}\ \boldsymbol{\varepsilon}$, and $\Gamma^{\varphi} \vdash t_2 : T^{q}\ \boldsymbol{\varepsilon}$, and:

$$\forall C : (\Gamma^{\varphi}; T^{q}\ \boldsymbol{\varepsilon}) \Rrightarrow (\varnothing; \mathsf{Unit}^{\varnothing}\ \boldsymbol{\varnothing}).\ C[t_1]\downarrow \Longleftrightarrow C[\,t_2\,]\downarrow .$$

We write $t\downarrow$ to mean term t terminates, if $\varnothing \mid t \longrightarrow_{\text{v}}^{*} \sigma \mid v$ for some value v and final store σ.

The above definition is standard [Ahmed et al. 2009] and defines a partial program equivalence. However, since we focus on a total fragment of the $\lambda^{*}_{\varepsilon}$-calculus here, program termination can not be used as an observer for program equivalence. We will thus rely on the following refined version of contextual equivalence using Boolean contexts:

$$\begin{gathered}\forall C : (\Gamma^{\varphi}; T^{q}\ \boldsymbol{\varepsilon}) \Rrightarrow (\varnothing; \mathsf{Bool}^{\varnothing}\ \boldsymbol{\varnothing}).\ \exists\, \sigma, \sigma', v. \\ \varnothing \mid C[t_1] \longrightarrow_{\text{v}}^{*} \sigma \mid v \ \wedge\ \varnothing \mid C[t_2] \longrightarrow_{\text{v}}^{*} \sigma' \mid v.\end{gathered}$$

That is to say, we consider two terms contextually equivalent if they yield the same answer value in all Boolean contexts.

7.3 The Model

Following other prior works [Ahmed 2004; Benton et al. 2007; Thamsborg and Birkedal 2011], we apply Kripke logical relations to the $\lambda^{*}_{\varepsilon}$-calculus. Our logical relations are indexed by types and store layouts via *worlds*. This allows us to interpret Ref B as an allocated location that holds values of type B. The invariant that all allocated locations hold well-typed values with respect to the world must hold in the pre-state and be re-established in the post-state of a computation. The world may grow as more locations may be allocated. It is important that this invariant must hold in future worlds, which is commonly referred as *monotonicity*.

Considering the restriction to first-order references here, our store layouts are always "flat", *i.e.*, free of cycles. The notion of world for the $\lambda^{*}_{\varepsilon}$-calculus is defined in the following:

Definition 7.2 (World). A world W is a triple (L_1, L_2, f), where

- L_1 and L_2 are finite sets of locations.
- $f \subseteq (L_1 \times L_2)$ is a partial bijection.

A world is meant to define relational stores. The partial bijection captures the fact that a relation holds under permutation of locations.

If $\mathrm{W} = (L_1, L_2, f)$ is a world, we refer to its components as follows:

$$\begin{aligned}\mathrm{W}(\ell_1, \ell_2) &= \begin{cases}(\ell_1, \ell_2) \in f & \text{when defined} \\ \varnothing & \text{otherwise}\end{cases} \\ \mathrm{dom}_1(\mathrm{W}) &= L_1 \\ \mathrm{dom}_2(\mathrm{W}) &= L_2\end{aligned}$$

If W and W′ are worlds, such that $\mathrm{dom}_1(\mathrm{W}) \cap \mathrm{dom}_1(\mathrm{W'}) = \mathrm{dom}_2(\mathrm{W}) \cap \mathrm{dom}_2(\mathrm{W'}) = \varnothing$, then W and W′ are called disjoint, and we write W; W′ to mean extending W with a disjoint world W′. Let σ_1 and σ_2 be two stores. We write $(\sigma_1, \sigma_2) : \mathrm{W}$ to mean $\mathrm{W} = (\mathrm{dom}(\sigma_1), \mathrm{dom}(\sigma_2), f)$.

Interpretation of Value Reachability

$$\mathrm{locs}(\omega) = \varnothing \qquad \mathrm{locs}(\mathrm{unit}) = \varnothing \qquad \mathrm{locs}(c) = \varnothing \qquad \mathrm{locs}(\ell) = \{\, \ell \,\}$$

$$\mathrm{locs}(\lambda x.t) = \{\ell \mid \ell \in \mathrm{fv}(t) \land \ell \in \mathrm{Loc}\}$$

Interpretation of Reachability Qualifiers

$$\mathrm{locs}(q) \overset{\mathrm{def}}{=} \{\, \ell \mid \ell \in q \land \ell \in \mathrm{Loc}\}$$

Reachability Predicates

$$v \rightsquigarrow^{\sigma} L \overset{\mathrm{def}}{=} (\mathrm{dom}(\sigma) \cap \mathrm{locs}(v)) \subseteq L$$

Fig. 13. Interpretation of reachability qualifiers.

Our world definition allows us to specify that the domains of two relational stores may grow during a computation, but does not cover store operations, which is important when proving the soundness of equational rules. Like prior works (*e.g.*, [Benton et al. 2007; Thamsborg and Birkedal 2011]), we use effects as a refinement for the definition of world. The notation $\boldsymbol{\varepsilon}$ denotes read/write effects, and ω means allocation occurs during a computation. Local reasoning is enabled by reachability qualifiers and read/write effects, meaning that what is preserved during an effectful computation are the locations that are *not* mentioned in the read/write effects. This is a common technique used in reasoning about frames in Hoare-style logics, *e.g.*, separation logic [Reynolds 2002]. This treatment is also applicable to our refined effect system (Sec. 6), where framing is achieved through write effects – an established technique in Dafny [Leino 2010] and region logics [Banerjee et al. 2013; Bao et al. 2015]. In this case, a frame indirectly describes the locations that a computation may not change [Borgida et al. 1995]. Framing allows the proof to carry properties of effectful terms, such as function applications, since properties that are true for unchanged locations will remain valid [Bao et al. 2018].

7.4 Interpretation of Reachability

In the λ^*_ε-calculus, reachability qualifiers are used to specify desired separation or permissible overlapping of reachable locations from a function's argument and its body. Fig. 13 shows the interpretation of reachability qualifiers. As in the λ^*_ε calculus, values cannot be cyclic, we axiomatize the definition of reachability, without proving termination. Here, we assume free variables are already substituted with values.

We use $\mathrm{locs}(v)$ to define the set of locations that are reachable from a given value v. Base type values, *i.e.*, ω of type Alloc, unit of type Unit, and other constants c of other base types B, do not reach any store locations. Thus, they reach the empty set of locations. A location ℓ can only reach itself. Thus, its reachable set is the singleton set $\{\ell\}$. The set of locations that are reachable from a function value $\lambda x.t$ are the set of the locations appearing in the function body.

We overload the function locs, and write $\mathrm{locs}(q)$ to mean the set of locations reachable from qualifier q, which are the set of the locations appeared in q. A bound variable may appear in q, and serves as a placeholder to specify the set of locations that a function's return value may reach. See Sec. 7.5 for details. The notation $v \rightsquigarrow^{\sigma} L$ is a predicate that asserts the set of locations that are reachable from v in store σ is a subset of L, where L is a set of locations.

7.5 Binary Logical Relations for λ^*_ε

This section presents the definition of binary logical relations for λ^*_ε. Following the approach of Timany et al. [2022], we define the binary logical relation for logical equivalence in two steps:

Value Interpretation of Types and Terms $\boxed{\lambda^*_\varepsilon}$

$$
\begin{aligned}
\mathcal{V}[[\mathsf{Alloc}]]^\gamma\ & \{(\mathrm{W}, \omega, \omega)\} \\
\mathcal{V}[[\mathsf{Unit}]]^\gamma &= \{(\mathrm{W}, \mathrm{unit}, \mathrm{unit})\} \\
\mathcal{V}[[\mathsf{Bool}]]^\gamma &= \{(\mathrm{W}, v, v) \mid v = \mathrm{true} \lor v = \mathrm{false}\} \\
\mathcal{V}[[\mathsf{Ref\ B}]]^\gamma &= \{(\mathrm{W}, \ell_1, \ell_2) \mid \forall\, \sigma_1, \sigma_2.\ (\sigma_1, \sigma_2) : \mathrm{W} \land \ell_1 \in \mathrm{dom}(\sigma_1) \land \ell_2 \in \mathrm{dom}(\sigma_2) \land \mathrm{W}(\ell_1, \ell_2) \land \\
&\quad (\mathrm{W}, \sigma_1(\ell_1), \sigma_2(\ell_2)) \in \mathcal{V}[[\mathrm{B}]]^\gamma\}
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{V}[[(x : T^p) \to^{\boldsymbol{\varepsilon}} U^r]]^\gamma &= \{(\mathrm{W}, \lambda x.t_1, \lambda x.t_2) \mid \mathrm{locs}(\lambda x.t_1) \subseteq \mathrm{dom}_1(\mathrm{W}) \land \mathrm{locs}(\lambda x.t_2) \subseteq \mathrm{dom}_2(\mathrm{W}) \land \\
&\quad (\forall v_1, v_2, \mathrm{W}', \sigma_1, \sigma_2.(\sigma_1, \sigma_2) : \mathrm{W};\mathrm{W}' \Rightarrow (\mathrm{W};\mathrm{W}', v_1, v_2) \in \mathcal{V}[[T]]^\gamma \Rightarrow \\
&\qquad \mathrm{locs}(\lambda x.t_1) \cap \mathrm{locs}(v_1) \subseteq \mathrm{locs}(\gamma_1(p)) \Rightarrow \mathrm{locs}(\lambda x.t_2) \cap \mathrm{locs}(v_2) \subseteq \mathrm{locs}(\gamma_2(p)) \Rightarrow \\
&\qquad \exists\, \mathrm{W}'', \sigma_1', \sigma_2', v_1', v_2'.\ \sigma_1 \mid t_1[x \mapsto v_1] \longrightarrow^*_{\mathrm{v}} \sigma_1' \mid v_1' \land \sigma_2 \mid t_2[x \mapsto v_2] \longrightarrow^*_{\mathrm{v}} \sigma_2' \mid v_2' \land \\
&\qquad\quad (\sigma_1', \sigma_2') : \mathrm{W};\mathrm{W}';\mathrm{W}'' \land (\mathrm{W};\mathrm{W}';\mathrm{W}'', v_1', v_2') \in \mathcal{V}[[U]]^\gamma \land \\
&\qquad\quad (x \in r \Rightarrow v_1' \leadsto^{\sigma_1} (\mathrm{locs}(\gamma_1(r)) \cap \mathrm{locs}(\lambda x.t_1) \cup \mathrm{locs}(v_1)) \land \\
&\qquad\qquad\quad v_2' \leadsto^{\sigma_2} (\mathrm{locs}(\gamma_2(r)) \cap \mathrm{locs}(\lambda x.t_2) \cup \mathrm{locs}(v_2))) \land \\
&\qquad\quad (x \notin r \Rightarrow v_1' \leadsto^{\sigma_1} (\mathrm{locs}(\gamma_1(r)) \cap \mathrm{locs}(\lambda x.t_1)) \land \\
&\qquad\qquad\quad v_2' \leadsto^{\sigma_2} (\mathrm{locs}(\gamma_2(r)) \cap \mathrm{locs}(\lambda x.t_2))) \land \\
&\qquad\quad (x \in \boldsymbol{\varepsilon} \Rightarrow \sigma_1 \hookrightarrow^{\mathrm{locs}(\gamma_1(\boldsymbol{\varepsilon} \rhd p))} \sigma_1' \land \sigma_2 \hookrightarrow^{\mathrm{locs}(\gamma_2(\boldsymbol{\varepsilon} \rhd p))} \sigma_2') \land \\
&\qquad\quad (x \notin \boldsymbol{\varepsilon} \Rightarrow \sigma_1 \hookrightarrow^{\mathrm{locs}(\gamma_1(\boldsymbol{\varepsilon}))} \sigma_1' \land \sigma_2 \hookrightarrow^{\mathrm{locs}(\gamma_2(\boldsymbol{\varepsilon}))} \sigma_2'))\} \\
\sigma \hookrightarrow^{\boldsymbol{\varepsilon}} \sigma' &\stackrel{\mathrm{def}}{=} \forall l \in \mathrm{dom}(\sigma).\sigma(l) = \sigma'(l) \lor l \in \boldsymbol{\varepsilon}
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{E}[[T^q\ \boldsymbol{\varepsilon}]]^\gamma_\varphi &= \{(\mathrm{W}, t_1, t_2) \mid \forall\, \sigma_1, \sigma_2.(\sigma_1, \sigma_2) : \mathrm{W} \land \exists\, \mathrm{W}', \sigma_1', \sigma_2', v_1, v_2.\ t_1 \mid \sigma_1 \longrightarrow^*_{\mathrm{v}} v_1 \mid \sigma_1' \land \\
&\quad t_2 \mid \sigma_2 \longrightarrow^*_{\mathrm{v}} v_2 \mid \sigma_2' \land (\sigma_1', \sigma_2') : \mathrm{W};\mathrm{W}' \land (\mathrm{W};\mathrm{W}', v_1, v_2) \in \mathcal{V}[[T]]^\gamma \land \\
&\quad v_1 \leadsto^{\sigma_1} (\mathrm{locs}(\gamma_1(\varphi \cap q))) \land v_2 \leadsto^{\sigma_2} (\mathrm{locs}(\gamma_2(\varphi \cap q))) \land \\
&\quad \sigma_1 \hookrightarrow^{\mathrm{locs}(\gamma_1(\boldsymbol{\varepsilon}))} \sigma_1' \land \sigma_2 \hookrightarrow^{\mathrm{locs}(\gamma_2(\boldsymbol{\varepsilon}))} \sigma_2'\}
\end{aligned}
$$

Fig. 14. Binary value and term interpretation for the λ^*_ε-calculus.

(1) We define binary interpretations on pairs of closed values, and pairs of closed terms.
(2) We define the logical equivalence relation on open terms, $\Gamma^\varphi \models t_1 \approx_{\log} t_2 : T^q\ \boldsymbol{\varepsilon}$, by lifting the value and term relations to open terms using a closing substitution.

The λ^*_ε-calculus has a dependent type system, where types may mention term variables. We could either define logical relations indexed by two types with variants on qualifiers (*i.e.*, the choice of locations after closing substitution); or indexed by a type where performing closing substitution to qualifiers in the definition. Here, we choose the latter. Thus, a relational value substitution is a parameter of the definition of logical relations.

The relational value substitution has to satisfy the context interpretation. We define the interpretation of typing contexts:

$$
\begin{aligned}
G[[\varnothing^\varphi]] &= \varnothing \\
G[[(\Gamma, x : T^q)^\varphi]] &= \{(\mathrm{W}, \gamma; (x \mapsto (v_1, v_2))) \mid (\mathrm{W}, \gamma) \in G[[\Gamma^\varphi]] \land \varphi \subseteq \mathrm{dom}(\Gamma) \land q \subseteq \mathrm{dom}(\Gamma) \land \\
&\quad (\mathrm{W}, v_1, v_2) \in \mathcal{V}[[T]]^\gamma \land \\
&\quad (\forall\, q, q'.\ q \subseteq \varphi \land q' \subseteq \varphi \land \Rightarrow \\
&\qquad (\mathrm{locs}(\gamma_1(q*)) \cap \mathrm{locs}(\gamma_1(q'*)) \subseteq \mathrm{locs}(\gamma_1(q* \cap q'*)) \land \\
&\qquad \mathrm{locs}(\gamma_2(q*)) \cap \mathrm{locs}(\gamma_2(q'*)) \subseteq \mathrm{locs}(\gamma_2(q* \cap q'*))))\}
\end{aligned}
$$

In the above definition, γ ranges over relational value substitutions that are finite maps from variables x to pairs of values (v_1, v_2). If $\gamma(x) = (v_1, v_2)$, then $\gamma_1(x)$ denotes v_1 and $\gamma_2(x)$ denotes v_2. We write $\gamma_1(q)$ and $\gamma_2(q)$ to mean substituting the free variables in q with respect to the relational value substitution γ.

The Binary Value Interpretation. The definition of binary value interpretation of types is shown in Fig. 14. The relational interpretation of type T, written as $\mathcal{V}[[T]]^\gamma$, is a set of tuples of form

(W, v_1, v_2), where v_1 and v_2 are values, and W is a world. We say v_1 and v_2 are related at type T with respect to W.

Ground Types. We restrict the base types B to Alloc, Unit and Bool to streamline the presentation. A pair of allocation capabilities (ω, ω) are related at type Alloc. A pair of unit values $(\mathrm{unit}, \mathrm{unit})$ are related at type Unit. A pair of boolean values are related if they are both true or false. A pair of locations (ℓ_1, ℓ_2) are related if they are in the domain of the relational store with respect to W, $((\sigma_1, \sigma_2) : \mathrm{W})$, such that $\mathrm{W}(\ell_1, \ell_2)$. It means that a pair of related locations store related values.

Function Types. Two λ terms, $\lambda x.t_1$ and $\lambda x.t_2$, are related at type $T^p \to^{\varepsilon} U^r$ with respect to world W, meaning that it satisfies the following conditions:

- The set of locations reachable from the two λ terms are well-formed with respect to the world, *i.e.*, $\mathrm{locs}(\lambda x.t_1) \subseteq \mathrm{dom}_1(\mathrm{W})$ and $\mathrm{locs}(\lambda x.t_2) \subseteq \mathrm{dom}_2(\mathrm{W})$.
- The arguments are allowed if
  - the arguments v_1 and v_2 are related at type T with respect $\mathrm{W}; \mathrm{W}'$, for all W'; and
  - the overlapping locations reachable from the functions and their arguments are permissible by the argument's qualifier p, *i.e.*, $\mathrm{locs}(\lambda x.t_1) \cap \mathrm{locs}(v_1) \subseteq \mathrm{locs}(\gamma_1(p))$ and $\mathrm{locs}(\lambda x.t_2) \cap \mathrm{locs}(v_2) \subseteq \mathrm{locs}(\gamma_2(p))$.
- After substitution, the two terms $t_1[x \mapsto v_1]$ and $t_2[x \mapsto v_2]$ are reduced to some values v_1' and v_2' with some final stores σ_1' and σ_2'.
- σ_1' and σ_2' are related with respect to world $\mathrm{W}; \mathrm{W}'; \mathrm{W}''$, for some W'', *i.e.*, $(\sigma_1', \sigma_2') : \mathrm{W}; \mathrm{W}'; \mathrm{W}''$.
- v_1' and v_2' are related at type U with respect to world $\mathrm{W}; \mathrm{W}'; \mathrm{W}''$.
- If the return value's qualifier r depends on the argument (*i.e.*, $x \in r$), then the locations reachable from v_1' and v_2' are subsets of those reachable both from the function and r, plus those reachable from the arguments; otherwise (*i.e.*, $x \notin r$), they are just subset of those reachable both from the function and r.
- If a bound variable x appears in the effect ε, meaning the function body may modify the argument, then the effect will include the qualifier that may reach the value of function argument p; otherwise it is just ε.

The Binary Term Interpretation. Two related terms, t_1 and t_2, are defined based on the relation of their computational behaviors, *i.e.*, returned values, reachability qualifiers and effects, which is defined by $\mathcal{E}[[T\ \varepsilon]]^{\gamma}_{\varphi}$. It means for all related stores with respect to world, $(\sigma_1, \sigma_2) : \mathrm{W}$, if

- t_1 is evaluated to some value v_1 with some final store σ_1';
- t_2 is evaluated to some value v_2 with some final store σ_2';
- v_1 and v_2 are related at type T with respect to world $\mathrm{W}; \mathrm{W}'$ for some W';
- σ_1' and σ_2' are related with respect to $\mathrm{W}; \mathrm{W}'$.
- The locations reachable from the values in the domain of pre-stores are subset of those reachable from $\gamma_1(\varphi \cap q)$ and $\gamma_2(\varphi \cap q)$ for each of the term.
- The effect captures what may be modified in the pre-state store.

Note that we interpret the function body (after substitution) and other terms separately, which allows us to provide more precise reasoning in the logical relations of function types.

Now, we define the binary logical relation for logical equivalence $\Gamma^{\varphi} \models t_1 \approx_{\log} t_2 : T^q\ \varepsilon$ as follows:

Definition 7.3.

$$\Gamma^{\varphi} \models t_1 \approx_{\log} t_2 : T^q\ \varepsilon \overset{\mathrm{def}}{=} \forall(\gamma, \mathrm{W}) \in G[[\Gamma^{\varphi}]].(\mathrm{W}, \gamma_1(t_1), \gamma_2(t_2)) \in \mathcal{E}[[T^q\ \varepsilon]]^{\gamma}_{\varphi}$$

where $\gamma_1(t)$ and $\gamma_2(t)$ means substitutions of the free variable in t_1 and t_2 with respect to the relational value substitution γ. Closing substitution over qualifiers and effects is performed in the definition of logical relations (Fig. 14).

7.6 Metatheory

This section discusses several key lemmas used in the proof of compatibility lemmas (Sec. 7.7) and soundness of equational rules (Sec. 7.9).

7.6.1 World Extension and Relational Stores.

Lemma 7.4 (Relational Store Update). *If* $(\sigma_1, \sigma_2) : W$, *and* $(W, \ell_1, \ell_2) \in \mathcal{V}[[\text{Ref B}]]^{\gamma}$, *and* $(W, v_1, v_2) \in \mathcal{V}[[\text{B}]]^{\gamma}$, *then* $(\sigma_1[\ell_1 \mapsto v_1], \sigma_2[\ell_2 \mapsto v_2]) : W$.

Proof. By definition of relational stores. □

Lemma 7.5 (Relational Store Extension). *If* $(\sigma_1, \sigma_2) : W$, *and* $(W, v_1, v_2) \in \mathcal{V}[[\text{B}]]^{\gamma}$, *then* $(\sigma_1; (\ell_1 : v_1), \sigma_2; \ell_2 : v_2) : W; (\ell_1, \ell_2, (\ell_1, \ell_2) \in f)$, *where* $\ell_1 \notin \text{dom}(\sigma_1)$ *and* $\ell_2 \notin \text{dom}(\sigma_2)$.

Proof. By definition of relational stores. □

Lemma 7.6 (Logical Relation Closed Under Relational Value Substitution Extension). *If* T *is closed under* Γ^{φ}, *and* $(W, \gamma) \in G[[\Gamma^{\varphi}]]$, *then* $(W, v_1, v_2) \in \mathcal{V}[[T]]^{\gamma}$ *if and only if* $(W, v_1, v_2) \in \mathcal{V}[[T]]^{\gamma;\gamma'}$, *for all* γ'.

Proof. By induction on type T and the constructs of values v_1 and v_2. □

Lemma 7.7 (Logical Relation Closed Under World Extension). *If* $(W, v_1, v_2) \in \mathcal{V}[[T]]^{\gamma}$, *then for all* W', $(W; W', v_1, v_2) \in \mathcal{V}[[T]]^{\gamma}$.

Proof. By induction on type T and the constructs of values v_1 and v_2. □

7.6.2 Well-formedness.

Lemma 7.8 (Well-formed value interpretation). *Let* $(W, \gamma) \in G[[\Gamma^{\varphi}]]$. *If* $(W, v_1, v_2) \in \mathcal{V}[[T]]^{\gamma}$, *then* $locs(v_1) \subseteq \text{dom}_1(W)$ *and* $locs(v_2) \subseteq \text{dom}_2(W)$.

Proof. By induction on type T and the constructs of value v_1 and v_2. □

Lemma 7.9 (Well-formed Typing context interpretation). *Let* $(W, \gamma) \in G[[\Gamma^{\varphi}]]$, *then for all* $q \subseteq \varphi$, $locs(\gamma_1(q)) \subseteq \text{dom}_1(W)$ *and* $locs(\gamma_2(q)) \subseteq \text{dom}_2(W)$.

Proof. By definition of the typing context interpretation and Lemma 7.8. □

Lemma 7.10. *Let* $(W, \gamma) \in G[[\Gamma^{\varphi}]]$, *then* $\text{dom}(\gamma_1) = \text{dom}(\gamma_2) = \text{dom}(\Gamma)$, *and* $\text{dom}(\Gamma)*$.

Proof. Immediately by the definition of typing context interpretation and the definition of saturation in Fig. 3. □

7.6.3 Semantic Typing Context.

Lemma 7.11 (Semantic Typing Context Tighten). *If* $(W, \gamma,) \in G[[\Gamma^{\varphi}]]$, *then for all* $p \subseteq \varphi$, $(W, \gamma) \in G[[\Gamma^{p}]]$.

Proof. By the definition of typing context interpretation. □

Lemma 7.12 (Semantic Typing Context Extension 1). *If* $(W, \gamma) \in G[[\Gamma^{\varphi}]]$, *and* $q \subseteq \text{dom}(\Gamma)$, *and* $(W, v_1, v_2) \in \mathcal{V}[[T]]^{\gamma}$, *and* $locs(\gamma_1(\varphi)) \cap locs(v_1) \subseteq locs(\gamma_1(q))$, *and* $locs(\gamma_2(\varphi)) \cap locs(v_2) \subseteq locs(\gamma_2(q))$, *then* $(W, \gamma; (x \mapsto (v_1, v_2))) \in G[[(\Gamma, x : T^{q})^{\varphi, x}]]$

Proof. By typing context interpretation and Lemma 7.6. □

Lemma 7.13 (Semantic Typing Context Extension 2). *If* $(W, \gamma) \in G[\![\Gamma^{\varphi}]\!]$*, and* $(W; W', v_1, v_2) \in \mathcal{V}[\![T]\!]^{\gamma}$*, and* $locs(\gamma_1(q)) \cap locs(v_1) \subseteq locs(\gamma_1(p))$*, and* $locs(\gamma_2(q)) \cap locs(v_2) \subseteq locs(\gamma_2(p))$*, and* $q \subseteq \varphi$*, then* $(W; W', \gamma; (x \mapsto (v_1, v_2))) \in G[\![(\Gamma, x : T^{p})^{q,x}]\!]$.

Proof. By typing context interpretation, Lemma 7.6, Lemma 7.7 and Lemma 7.11. □

7.6.4 *Reachability Qualifiers.*

Lemma 7.14. *For all* σ*,* p *and* q*,* $\omega \rightsquigarrow^{\sigma} locs(p \cap q)$.

Proof. Immediate by the definition in Fig. 13. □

Lemma 7.15. *For all* σ*,* p *and* q*,* $\mathsf{unit} \rightsquigarrow^{\sigma} locs(p \cap q)$.

Proof. Immediate by the definition in Fig. 13. □

Lemma 7.16. *For all* σ*,* b*,* p *and* q*,* $b \rightsquigarrow^{\sigma} locs(p \cap q)$*, where* b *is true or false.*

Proof. Immediate by the definition in Fig. 13. □

Lemma 7.17. *For all* σ*,* ℓ*,* p *and* q*,* $\ell \rightsquigarrow^{\sigma} locs(p \cap q)$*, where* $\ell \notin \mathrm{dom}(\sigma)$.

Proof. Immediate by the definition in Fig. 13. □

Lemma 7.18. *If* $(\Gamma, x : T^{p})^{q,x} \vdash t : U^{r}$*, then for all* $(W, \gamma) \in G[\![(\Gamma, x : T^{p})^{q,x}]\!]$*,* $\gamma_1(\lambda x.t) \rightsquigarrow^{\mathrm{dom}_1(W)} locs(\gamma_1(q))$ *and* $\gamma_2(\lambda x.t) \rightsquigarrow^{\mathrm{dom}_2(W)} locs(\gamma_2(q))$.

Proof. By the syntactic structure that ensures q contains all the free variables in t, it is obvious that after substituting free variables with values, the conclusions hold. □

7.6.5 *Effects.*

Here we introduce several notations to streamline the presentation.

Let σ be a store. We write $\sigma = \sigma_1 * \sigma_2$ (for some σ_1 and σ_2) to denote that store σ can be split into two disjoint parts σ_1 and σ_2.

Let σ be a store and L be a set of locations. We write $(\sigma{\downharpoonright}L)$ to mean localizing a partial store with respect to L, meaning $\mathrm{dom}((\sigma{\downharpoonright}L)) = \mathrm{dom}(\sigma) \cap L \ \wedge\ \forall\, \ell \in \mathrm{dom}((\sigma{\downharpoonright}L)).(\sigma{\downharpoonright}L)(\ell) = \sigma(\ell)$.

Lemma 7.19 (Read/Write Effects). *If* $\ell \in \mathrm{dom}(\sigma)$*, and* $\ell \rightsquigarrow^{\sigma} locs(p \cap q)$*, then* $\sigma \hookrightarrow^{locs(q)} \sigma[\ell \mapsto v]$.

Proof. By Lemma 7.17 and interpretation of effects. □

Lemma 7.20 (No Effects). $\sigma \hookrightarrow^{\varnothing} \sigma$.

Proof. Immediate by the definition of effects. □

Lemma 7.21 (SubEffects). *If* $locs(\varepsilon_1) \subseteq locs(\varepsilon_2)$*, and* $\sigma \hookrightarrow^{locs(\varepsilon_1)} \sigma'$*, then* $\sigma \hookrightarrow^{locs(\varepsilon_2)} \sigma'$.

Proof. By the interpretation of effects. □

Lemma 7.22 (Effects Composition). *If* $\sigma \hookrightarrow^{locs(\varepsilon_1 *)} \sigma'$*, and* $\sigma' \hookrightarrow^{locs((\varepsilon_2 \rhd \varepsilon_3)*)} \sigma''$*, and* $\varepsilon_2 * \cap \varepsilon_3 * = \varnothing$*, and* $locs(\varepsilon_2 *) \subseteq \mathrm{dom}(\sigma)$*, and* $locs(\varepsilon_3 *) \cap \mathrm{dom}(\sigma) = \varnothing$*. then* $\sigma \hookrightarrow^{locs((\varepsilon_1 \rhd \varepsilon_2)*)} \sigma''$

Proof. By the interpretation of effects. □

Lemma 7.23 (Framing). *If* $\sigma \hookrightarrow^{locs(\varepsilon)} \sigma'$*, then* $\sigma{\downarrow}(\mathrm{dom}(\sigma) - locs(\varepsilon *)) = \sigma'{\downarrow}(\mathrm{dom}(\sigma) - locs(\varepsilon *))$

Proof. By the interpretation of observable effects: the set of locations that may be written in the reduction of t must be in $\boldsymbol{\varepsilon}$. Thus, the values stored in the locations σ, but are separate from $\boldsymbol{\varepsilon}\ast$ must be preserved. □

Lemma 7.24 (Effect Separation). *If* $\sigma \mid t_1 \longrightarrow_{\mathrm{v}}^{*} \sigma' \mid v_1$, *and* $\sigma \mid t_2 \longrightarrow_{\mathrm{v}}^{*} \sigma'' \mid v_2$, *and* $\sigma \hookrightarrow^{locs(\boldsymbol{\varepsilon}_1)} \sigma'$, *and* $\sigma \hookrightarrow^{locs(\boldsymbol{\varepsilon}_2)} \sigma''$, *and* $\boldsymbol{\varepsilon}_1\ast \cap \boldsymbol{\varepsilon}_2\ast = \emptyset$, *then* $\sigma' \downarrow (\mathrm{dom}(\sigma) - locs(\boldsymbol{\varepsilon}_1\ast) - locs(\boldsymbol{\varepsilon}_2\ast)) = \sigma'' \downarrow (\mathrm{dom}(\sigma) - locs(\boldsymbol{\varepsilon}_1\ast) - locs(\boldsymbol{\varepsilon}_2\ast))$.

Proof. Let $\sigma_1 = \sigma \downarrow \mathrm{locs}(\boldsymbol{\varepsilon}_1\ast)$, and $\sigma_2 = \sigma \downarrow \mathrm{locs}(\boldsymbol{\varepsilon}_2\ast)$, and $\sigma_3 = \sigma \downarrow (\mathrm{dom}(\sigma) - \mathrm{locs}(\boldsymbol{\varepsilon}_1\ast) - \mathrm{locs}(\boldsymbol{\varepsilon}_2\ast))$, We know that $\sigma = \sigma_1 \ast \sigma_2 \ast \sigma_3$, as $\boldsymbol{\varepsilon}_1\ast \cap \boldsymbol{\varepsilon}_2\ast = \emptyset$.

By $\sigma \hookrightarrow^{\varepsilon_1} \sigma'$ and Lemma 7.23, we know $\sigma' = \sigma_1' \ast \sigma_2 \ast \sigma_3 \ast \sigma_{fr1}$, for some σ_1', where $\sigma_{fr1} \ast \sigma$.

By $\sigma \hookrightarrow^{\varepsilon_2} \sigma''$ and Lemma 7.23, we know $\sigma'' = \sigma_1 \ast \sigma_2' \ast \sigma_3 \ast \sigma_{fr2}$, for some σ_2', where $\sigma_{fr2} \ast \sigma$.

Then $\sigma' \downarrow (\mathrm{dom}(\sigma) - \mathrm{locs}(\boldsymbol{\varepsilon}_1\ast) - \mathrm{locs}(\boldsymbol{\varepsilon}_2\ast)) = \sigma_3$, and $\sigma'' \downarrow (\mathrm{dom}(\sigma) - \mathrm{locs}(\boldsymbol{\varepsilon}_1\ast) - \mathrm{locs}(\boldsymbol{\varepsilon}_2\ast)) = \sigma_3$. □

7.6.6 Other auxiliary lemmas.

Lemma 7.25 (Qualifier intersection distributes over locations). *Let* $(W, \gamma) \in G\llbracket \Gamma^{\varphi} \rrbracket$, *and* $(\sigma_1, \sigma_2) : W$. *For all* σ_1', σ_2' *and* W', *such that* $(\sigma_1', \sigma_2') : W; W'$ *if* $v_{f1} \rightsquigarrow^{\sigma_1} locs(\gamma_1(q_f))$, *and* $v_{f2} \rightsquigarrow^{\sigma_2} locs(\gamma_2(q_f))$, *and* $v_1 \rightsquigarrow^{\sigma_1'} locs(\gamma_1(p))$, *and* $v_2 \rightsquigarrow^{\sigma_2'} locs(\gamma_2(p))$, *and* $locs(v_{f1}) \subseteq \mathrm{dom}(\sigma_1')$, *and* $locs(v_{f2}) \subseteq \mathrm{dom}(\sigma_2')$, *then* $(locs(v_{f1}) \cap locs(v_1)) \subseteq locs(\gamma_1(p\ast \cap q_f\ast))$ *and* $(locs(v_{f2}) \cap locs(v_2)) \subseteq locs(\gamma_2(p\ast \cap q_f\ast))$.

Proof. By typing context interpretation, Lemma 7.9 and set theory. □

Lemma 7.26 (Semantic Function Abstraction). *Let* $(W, \gamma) \in G\llbracket \Gamma^{\varphi} \rrbracket$, $(\sigma_1, \sigma_2) : W$, *and* $\mathrm{dom}(\Gamma)\ast$. *For all* W', *if* $(W; W', v_1, v_2) \in \mathcal{V}\llbracket T \rrbracket^{\gamma}$, *and* $locs(\gamma_1(\lambda x.t_1)) \cap locs(v_1) \subseteq locs(\gamma_1(p))$, *and* $locs(\gamma_2(\lambda x.t_2)) \cap locs(v_2) \subseteq locs(\gamma_2(p))$, *and* $(W; W', \gamma; x \mapsto (v_1, v_2)) \in G\llbracket (\Gamma, x : T^{p})^{q,x} \rrbracket$ *implies that there exists* W'', *such that*
$(W; W'; W'', \gamma_1(t_1)[x \mapsto v_1], \gamma_2(t_2)[x \mapsto v_2]) \in \mathcal{E}\llbracket U^{r}\ \boldsymbol{\varepsilon} \rrbracket^{\gamma}_{q,x}$. *and* $p \subseteq q$, *then exists* W'', v_1', v_2', *such that*

(1) $\sigma_1 \mid \gamma_1(t_1)[x \mapsto v_1] \longrightarrow_{\mathrm{v}}^{*} \sigma_1' \mid v_2$
(2) $\sigma_2 \mid \gamma_2(t_2)[x \mapsto v_2] \longrightarrow_{\mathrm{v}}^{*} \sigma_2' \mid v_2'$
(3) $(\sigma_1', \sigma_2') : W; W'; W''$
(4) $(W; W'; W'', v_3, v_4) \in \mathcal{V}\llbracket \mathrm{U} \rrbracket^{\gamma}$
(5) $(x \in r \Rightarrow v_1' \rightsquigarrow^{\sigma_1'} (locs(\gamma_1(r)) \cap locs(\gamma_1(\lambda x.t_1)) \cup locs(v_1)) \wedge$
$v_2' \rightsquigarrow^{\sigma_2'} (locs(\gamma_2(r)) \cap locs(\gamma_2(\lambda x.t_2)) \cup locs(v_2)))$
(6) $(x \notin r \Rightarrow v_1' \rightsquigarrow^{\sigma_1'} (locs(\gamma_1(r)) \cap locs(\gamma_1(\lambda x.t_1))) \wedge v_2' \rightsquigarrow^{\sigma_2'} (locs(\gamma_2(r)) \cap locs(\gamma_2(\lambda x.t_2))))$

Proof. By Lemma 7.13, $(\mathrm{W}; \mathrm{W}', \gamma; (x \mapsto (v_1, v_2))) \in G\llbracket (\Gamma, x : T^{p})^{q,x} \rrbracket$. Thus, there exists W'', such that $(\mathrm{W}; \mathrm{W}'; \mathrm{W}'', \gamma_1(t_1)[x \mapsto v_1], \gamma_2(t_2)[x \mapsto v_2]) \in \mathcal{E}\llbracket U^{r}\ \boldsymbol{\varepsilon} \rrbracket^{\gamma}_{q,x}$, which can be used to prove (1) - (4). (5) and (6) can be proved by inspecting $x \in r$, Lemma 7.8 and Lemma 7.18. □

Lemma 7.27 (Semantic Application). *Let* $(W, \gamma) \in G\llbracket \Gamma^{\varphi} \rrbracket$. *If* $(W; W', \gamma_1(\lambda x.t_1), \gamma_2(\lambda x.t_2)) \in \mathcal{V}\llbracket \mathsf{T}^{p\ast \cap q\ast} \rightarrow^{\boldsymbol{\varepsilon}} U^{r} \rrbracket^{\gamma}$, *and* $\gamma_1(\lambda x.t_1) \rightsquigarrow^{\sigma_1} locs(\gamma_1(q))$, *and* $\gamma_2(\lambda x.t_2) \rightsquigarrow^{\sigma_2} locs(\gamma_2(q))$, *and* $(W; W'; W'', v_1, v_2) \in \mathcal{V}\llbracket T \rrbracket^{\gamma}$, *and* $v_1 \rightsquigarrow^{\mathrm{dom}_1(W;\, W')} locs(\gamma_1(p))$, *and* $v_2 \rightsquigarrow^{\mathrm{dom}_2(W;\, W')} locs(\gamma_2(p))$, *and* $r \subseteq \varphi, x$, *and* $\boldsymbol{\varepsilon} \subseteq q, x$, *and* $(\sigma_1, \sigma_2) : W; W'; W''$ *then there exists* v_2, v_2', σ_1', σ_2', W''', *such that*

(1) $\sigma_1 \mid \gamma_1(t_1)[x \mapsto v_1] \longrightarrow_{\mathrm{v}}^{*} \sigma_1' \mid v_2$;
(2) $\sigma_2 \mid \gamma_1(t_2)[x \mapsto v_2] \longrightarrow_{\mathrm{v}}^{*} \sigma_2' \mid v_2'$;
(3) $(\sigma_1', \sigma_2') : W; W'; W''; W'''$;

(4) $(W; W'; W''; W''', v_2, v_2') \in \mathcal{V}[[U]]^{\gamma}$;
(5) $x \in r \Rightarrow v_1' \rightsquigarrow^{\sigma_1} (locs(\gamma_1(r)) \cap locs(\gamma_1(\lambda x.t_1)) \cup locs(v_1)) \wedge$
$v_2' \rightsquigarrow^{\sigma_2} (locs(\gamma_2(r)) \cap locs(\gamma_2(\lambda x.t_2)) \cup locs(v_2))$
(6) $x \notin r \Rightarrow v_1' \rightsquigarrow^{\sigma_1} (locs(\gamma_1(r)) \cap locs(\gamma_1(\lambda x.t_1))) \wedge v_2' \rightsquigarrow^{\sigma_2} (locs(\gamma_2(r)) \cap locs(\gamma_2(\lambda x.t_2)))$

Proof. By Lemma 7.8, we know the following:

- $\mathrm{locs}(\gamma_1(\lambda x.t_1)) \subseteq \mathrm{dom}_1(\mathrm{W};\mathrm{W}')$;
- $\mathrm{locs}(\gamma_2(\lambda x.t_2)) \subseteq \mathrm{dom}_2(\mathrm{W};\mathrm{W}')$;
- $\mathrm{locs}(v_1) \subseteq \mathrm{dom}_1(\mathrm{W};\mathrm{W}';\mathrm{W}'')$;
- $\mathrm{locs}(v_2) \subseteq \mathrm{dom}_2(\mathrm{W};\mathrm{W}';\mathrm{W}'')$;

Then (1) - (4) can be proved by the assumption $(\mathrm{W};\mathrm{W}', \gamma_1(\lambda x.t_1), \gamma_2(\lambda x.t_2)) \in \mathcal{V}[[\mathrm{T}^p \to^{\varepsilon} U^r]]^{\gamma}$, and $(\mathrm{W};\mathrm{W}';\mathrm{W}'', v_1, v_2) \in \mathcal{V}[[T]]^{\gamma}$, and Lemma 7.25. (5) - (6) can be proved by inspecting $x \in r$. □

7.7 Compatibility Lemmas

The following compatibility lemmas show that the logical relations is *compatible* with all the constructs of the language [Pierce 2004].

Lemma 7.28 (Compatibility: Alloc). $\Gamma^{\varphi} \models \omega \approx_{log} \omega : Alloc^{\varnothing}\ \varnothing$

Proof. By the typing context interpretation, value interpretation in Fig. 14 and Lemma 7.14. □

Lemma 7.29 (Compatibility: Unit). $\Gamma^{\varphi} \models \mathsf{unit} \approx_{log} \mathsf{unit} : \mathsf{Unit}^{\varnothing}\ \varnothing$

Proof. By the typing context interpretation, value interpretation in Fig. 14 and Lemma 7.15. □

Lemma 7.30 (Compatibility: Bool). $\Gamma^{\varphi} \models true \approx_{log} true : Bool^{\varnothing}\ \varnothing$

Proof. By the typing context interpretation, value interpretation in Fig. 14 and Lemma 7.16. □

Lemma 7.31 (Compatibility: Bool). $\Gamma^{\varphi} \models false \approx_{log} false : Bool^{\varnothing}\ \varnothing$

Proof. By the typing context interpretation, value interpretation in Fig. 14 and Lemma 7.16. □

Lemma 7.32 (Compatibility: Variables). *If* $x : T^q \in \Gamma$ *and* $x \subseteq \varphi$, *then* $\Gamma^{\varphi} \models x \approx_{log} x : T^x\ \varnothing$

Proof. Immediate by the typing context interpretation in Fig. 14. □

Lemma 7.33 (Compatibility: λ). *If* $(\Gamma\ ,\ x : T^p)^{q,x} \models t_1 \approx_{log} t_2 : U^r\ \varepsilon$, $q \subseteq \varphi$, *then* $\Gamma^{\varphi} \models \lambda x.t_1 \approx_{log} \lambda x.t_2 : (x : T^p \to^{\varepsilon} U^r)^q\ \varnothing$.

Proof. Let $(\mathrm{W}, \gamma) \in G[[\Gamma]]$ and $(\sigma_1, \sigma_2) : \mathrm{W}$.
By definition of term interpretation, we need to show there exists W', σ', v_1 and v_2 such that:

(1) $\sigma_1 \mid \gamma_1(\lambda x.t_1) \longrightarrow_{\mathrm{v}}^{*} \sigma_1' \mid v_1$
(2) $\sigma_2 \mid \gamma_2(\lambda x.t_2) \longrightarrow_{\mathrm{v}}^{*} \sigma_2' \mid v_2$
(3) $(\sigma_1', \sigma_2') : \mathrm{W};\mathrm{W}'$
(4) $(\mathrm{W};\mathrm{W}', v_1, v_2) \in \mathcal{V}[[(x : T^p \to^{\varepsilon} U^r)]]^{\gamma}$
(5) $v_1 \rightsquigarrow^{\sigma_1} \mathrm{locs}(\gamma_1(\varphi \cap q))$
(6) $v_2 \rightsquigarrow^{\sigma_2} \mathrm{locs}(\gamma_2(\varphi \cap q))$
(7) $\sigma_1 \hookrightarrow^{\varnothing} \sigma_1'$
(8) $\sigma_2 \hookrightarrow^{\varnothing} \sigma_2'$

By reduction semantics, we pick $\mathrm{W} = \varnothing$, $v_1 = \lambda x.t_1$, $v_2 = \lambda x.t_2$, $\sigma_1' = \sigma_1$ and $\sigma_2' = \sigma_2$. Thus, (1)-(3) are discharged. (4) can be proved by Lemma 7.10 and Lemma 7.26. (5) and (6) can be proved by Lemma 7.18. (7) and (8) can be proved by Lemma 7.20. □

Lemma 7.34 (Compatibility : Allocation). *If* $\Gamma^{\varphi} \models t_1 \approx_{log} t_2 : \mathsf{Alloc}^{q}\ \boldsymbol{\varepsilon_1}$, *and* $\Gamma^{\varphi} \vdash t_3 \approx_{log} t_4 : B^{\varnothing}\ \boldsymbol{\varepsilon_2}$, *then* $\Gamma^{\varphi} \models \boldsymbol{ref}_{t_1}\ t_3 \approx_{log} \boldsymbol{ref}_{t_2}\ t_4\ : (\mathsf{Ref}\ B)^{\varnothing}\ \boldsymbol{\varepsilon_1} \rhd \boldsymbol{\varepsilon_2} \rhd q$.

Proof. Let $(\mathrm{W}, \gamma) \in G[[\Gamma]]$ and $(\sigma_1, \sigma_2) : \mathrm{W}$. By the first assumption, we know that there exists σ_1', σ_2', W', v_1 and v_2, such that

- $\gamma_1(t_1) \mid \sigma_1 \longrightarrow_{\mathrm{v}}^{*} v_1 \mid \sigma_1'$
- $\gamma_2(t_2) \mid \sigma_2 \longrightarrow_{\mathrm{v}}^{*} v_2 \mid \sigma_2'$
- $(\sigma_1', \sigma_2') : \mathrm{W}; \mathrm{W}'$
- $(\mathrm{W}; \mathrm{W}', v_1, v_2) \in \mathcal{V}[[\mathsf{Alloc}]]^{\gamma}$
- $v_1 \rightsquigarrow^{\sigma_1} \mathrm{locs}(\gamma_1(\varphi \cap q))$
- $v_2 \rightsquigarrow^{\sigma_2} \mathrm{locs}(\gamma_2(\varphi \cap q))$
- $\sigma_1 \hookrightarrow^{\mathrm{locs}(\gamma_1(\boldsymbol{\varepsilon_1}))} \sigma_1'$
- $\sigma_2 \hookrightarrow^{\mathrm{locs}(\gamma_2(\boldsymbol{\varepsilon_1}))} \sigma_2'$

By reduction semantics, we know $v_1 = v_2 = \omega$.
By the second assumption, we know that there exists σ_1'', σ_2'', W'', v_3 and v_4, such that

- $\gamma_1(t_3) \mid \sigma_1' \longrightarrow_{\mathrm{v}}^{*} v_3 \mid \sigma_1''$
- $\gamma_2(t_4) \mid \sigma_2' \longrightarrow_{\mathrm{v}}^{*} v_4 \mid \sigma_2''$
- $(\sigma_1'', \sigma_2'') : \mathrm{W}; \mathrm{W}'; \mathrm{W}''$
- $(\mathrm{W}; \mathrm{W}'; \mathrm{W}'', v_3, v_4) \in \mathcal{V}[[\mathrm{B}]]^{\gamma}$
- $v_3 \rightsquigarrow^{\sigma_1'} \mathrm{locs}(\gamma_1(\varphi \cap \varnothing))$
- $v_4 \rightsquigarrow^{\sigma_2'} \mathrm{locs}(\gamma_2(\varphi \cap \varnothing))$
- $\sigma_1' \hookrightarrow^{\mathrm{locs}(\gamma_1(\boldsymbol{\varepsilon_2}))} \sigma_1''$
- $\sigma_2' \hookrightarrow^{\mathrm{locs}(\gamma_2(\boldsymbol{\varepsilon_2}))} \sigma_2''$

By reduction semantics, we know

- $\boldsymbol{ref}_{\omega}\ v_3 \mid \sigma_1'' \longrightarrow_{\mathrm{v}}^{1} \ell_1 \mid \sigma_1''; (\ell_1 \mapsto v_3)$, where $\ell_1 \notin \mathrm{dom}(\sigma_1'')$
- $\boldsymbol{ref}_{\omega}\ v_4 \mid \sigma_2'' \longrightarrow_{\mathrm{v}}^{1} \ell_2 \mid \sigma_2''; (\ell_2 \mapsto v_4)$, where $\ell_2 \notin \mathrm{dom}(\sigma_2'')$

By Lemma 7.5, we know $(\sigma_1''; (\ell_1 \mapsto v_3), \sigma_2''; (\ell_2 \mapsto v_4)) : \mathrm{W}; \mathrm{W}'; \mathrm{W}''; ((\ell_1 \mapsto v_3), (\ell_2 \mapsto v_4), \{(\ell_1, \ell_2)\})$. The rest of the proof can be done by the definition of value interpretation, Lemma 7.22 and Lemma 7.17. □

Lemma 7.35 (Compatibility: Dereference (!)). *If* $\Gamma^{\varphi} \models t_1 \approx_{log} t_2 : (\mathsf{Ref}\ B)^{q}\ \boldsymbol{\varepsilon}$, *then* $\Gamma^{\varphi} \models !t_1 \approx_{log} !t_2 : B^{\varnothing}\ \boldsymbol{\varepsilon} \rhd \boldsymbol{q}$.

Proof. Let $(\mathrm{W}, \gamma) \in G[[\Gamma^{\varphi}]]$ and $(\sigma_1, \sigma_2) : \mathrm{W}$. By the assumption, $(\mathrm{W}, \gamma_1(t_1), \gamma_2(t_2)) \in \mathcal{E}[[\mathsf{Ref}\ B^{q}\ \boldsymbol{\varepsilon}]]_{\varphi}^{\gamma}$, and reduction semantics, we know there exists σ_1', σ_2', ℓ_1 and ℓ_2 such that

- $\sigma_1 \mid \gamma_1(t_1) \longrightarrow_{\mathrm{v}}^{*} \sigma_1' \mid \ell_1$
- $\sigma_2 \mid \gamma_2(t_2) \longrightarrow_{\mathrm{v}}^{*} \sigma_2' \mid \ell_2$
- $(\sigma_1', \sigma_2') : \mathrm{W}; \mathrm{W}'$
- $(\mathrm{W}; \mathrm{W}', \ell_1, \ell_2) \in \mathcal{V}[[\mathsf{Ref}\ B]]^{\gamma}$
- $\ell_1 \rightsquigarrow^{\sigma_1} \mathrm{locs}(\gamma_1(\varphi \cap q))$
- $\ell_2 \rightsquigarrow^{\sigma_2} \mathrm{locs}(\gamma_2(\varphi \cap q))$
- $\sigma_1 \hookrightarrow^{\mathrm{locs}(\gamma_1(\boldsymbol{\varepsilon}))} \sigma_1'$
- $\sigma_2 \hookrightarrow^{\mathrm{locs}(\gamma_2(\boldsymbol{\varepsilon}))} \sigma_2'$

We can finish the proof by reduction semantics, value interpretation, Lemma 7.16, Lemma 7.21, where we pick σ_1'' to be σ_1', σ_2'' to be σ', and W'' to be $\varnothing$.

□

Lemma 7.36 (Compatibility: Assignments (:=)). *If* $\Gamma^{\varphi} \models t_1 \approx_{log} t_2 : (\mathsf{Ref\ B})^{q}\ \varepsilon_1$, $\Gamma^{\varphi} \models t_3 \approx_{log} t_4 : \mathsf{B}^{\varnothing}\ \varepsilon_2$, *then* $\Gamma^{\varphi} \models t_1 := t_3 \approx_{log} t_2 := t_4 : \mathsf{Unit}^{\varnothing}\ \varepsilon_1 \rhd \varepsilon_2 \rhd q$.

Proof. Let $(\mathrm{W}, \gamma) \in G[[\Gamma^{\varphi}]]$ and $(\sigma_1, \sigma_2) : \mathrm{W}$. By the first assumption, we know that there exists σ_1', σ_2', W', v_1 and v_2 such that

- $\gamma_1(t_1) \mid \sigma_1 \longrightarrow_{\mathrm{v}}^{*} v_1 \mid \sigma_1'$
- $\gamma_2(t_2) \mid \sigma_2 \longrightarrow_{\mathrm{v}}^{*} v_2 \mid \sigma_2'$
- $(\sigma_1', \sigma_2') : \mathrm{W};\mathrm{W}'$
- $(\mathrm{W};\mathrm{W}', v_1, v_2) \in \mathcal{V}[[\mathsf{Ref\ B}]]^{\gamma}$
- $v_1 \rightsquigarrow^{\sigma_1} \mathrm{locs}(\gamma_1(\varphi \cap q))$
- $v_2 \rightsquigarrow^{\sigma_2} \mathrm{locs}(\gamma_2(\varphi \cap q))$
- $\sigma_1 \hookrightarrow^{\mathrm{locs}(\gamma_1(\varepsilon_1))} \sigma_1'$
- $\sigma_2 \hookrightarrow^{\mathrm{locs}(\gamma_2(\varepsilon_1))} \sigma_2'$

By the second assumption, we know that there exists σ_1'', σ_2'', W'', v_3 and v_4, such that

- $\gamma_1(t_3) \mid \sigma_1' \longrightarrow_{\mathrm{v}}^{*} v_3 \mid \sigma_1''$
- $\gamma_2(t_4) \mid \sigma_2' \longrightarrow_{\mathrm{v}}^{*} v_4 \mid \sigma_2''$
- $(\sigma_1'', \sigma_2'') : \mathrm{W};\mathrm{W}';\mathrm{W}''$
- $(\mathrm{W};\mathrm{W}';\mathrm{W}'', v_3, v_4) \in \mathcal{V}[[\mathsf{B}]]^{\gamma}$
- $v_3 \rightsquigarrow^{\sigma_1'} \mathrm{locs}(\gamma_1(\varphi \cap \varnothing))$
- $v_4 \rightsquigarrow^{\sigma_2'} \mathrm{locs}(\gamma_2(\varphi \cap \varnothing))$
- $\sigma_1' \hookrightarrow^{\mathrm{locs}(\gamma_1(\varepsilon_2))} \sigma_1''$
- $\sigma_2' \hookrightarrow^{\mathrm{locs}(\gamma_2(\varepsilon_2))} \sigma_2''$

Then the proof can be done by the reduction semantics, Lemma 7.4, value interpretation, Lemma 7.15, Lemma 7.19 and Lemma 7.22.

□

Lemma 7.37 (Compatibility: Applications (β)). . *If* $\Gamma^{\varphi} \models t_1 \approx_{log} t_2 : \left(x : T^{p* \cap q*} \to^{\varepsilon_3} U^{r}\right)^{q}\ \varepsilon_2$, *and* $\Gamma^{\varphi} \models t_3 \approx_{log} t_4 : T^{p}\ \varepsilon_1$, *and* $x \notin \mathrm{fv}(U)$, *and* $r \subseteq \varphi, x$, *and and* $\varepsilon_3 \subseteq \varphi, x$, *and* $\theta = [p/x]$, *then* $\Gamma^{\varphi} \models t_1\ t_3 \approx_{log} t_2\ t_4 : (U^{r}\ \varepsilon_1 \rhd \varepsilon_2 \rhd \varepsilon_3)\theta$.

Proof. The proof is done by the definition of term interpretation, Lemma 7.27 and Lemma 7.22. □

Lemma 7.38 (Compatibility: Let). *If* $\Gamma^{\varphi} \models t_1 \approx_{log} t_2 : S^{p}\ \varepsilon_1$, *and* $(\Gamma\,,\ x : S^{p* \cap \varphi*})^{\varphi, x} \models t_3 \approx_{log} t_4 : T^{q}\ \varepsilon_2$, *and* $\theta = [p/x]$ *and* $x \notin \mathrm{fv}\,(T)$, *then* $\Gamma^{\varphi} \models \textbf{let}\ x = t_1\ \textbf{in}\ t_3 \approx_{log} \textbf{let}\ x = t_2\ \textbf{in}\ t_4 : (T^{q}\ \varepsilon_1 \rhd \varepsilon_2)\theta$

Proof. Since the t-let is a combination of rules t-abs, t-app and weakening, the proof is analogous. □

Lemma 7.39 (Compatibility: Subtyping). *If* $\Gamma^{\varphi} \models t_1 \approx_{log} t_2 : S^{p}\ \varepsilon_1$ *and* $\Gamma \vdash S^{p}\ \varepsilon_1 <: T^{q}\ \varepsilon_2$ *and* $q, \varepsilon_2 \subseteq \varphi$, *then* $\Gamma^{\varphi} \models t_1 \approx_{log} t_2 : T^{q}\ \varepsilon_2$.

Proof. By induction on the subtyping derivation. □

7.8 The Fundamental Theorem and Soundness

Theorem 7.40 (Fundamental Property). *If* $\Gamma^{\varphi} \vdash t : T^{q}\ \varepsilon$, *then* $\Gamma^{\varphi} \models t \approx_{log} t : T^{q}\ \varepsilon$.

Proof. By induction on the derivation of $\Gamma^{\varphi} \vdash t : T^{q}\ \varepsilon$. Each case follows from the corresponding compatibility lemma. □

$$(\textsc{dce})\ \frac{\Gamma^{\varphi} \vdash t_1 : T_1^{q_1}\ \varepsilon_1 \qquad \Gamma^{\varphi} \vdash t_2 : T_2^{q_2}\ \varepsilon_2 \qquad t_1 \text{ terminates} \qquad \varepsilon_1 = \varnothing \text{ or } \omega}{\Gamma^{\varphi} \models \textbf{let } x = t_1 \textbf{ in } t_2 \approx_{\log} t_2 : T_2^{q_2}\,\varepsilon_2}$$

$$(\textsc{comm})\ \frac{\begin{array}{c}\Gamma^{\varphi} \vdash t_1 : T_1^{q_1}\ \varepsilon_1 \\ \Gamma^{\varphi} \vdash t_2 : T_2^{q_2}\ \varepsilon_2 \qquad \lceil \Gamma\,,\ x : T_1^{q_1 * \cap \varphi *},\ y : T_2^{q_2 * \cap (\varphi, x) *} \rceil^{\varphi, x, y} \vdash t : T^{q}\ \varepsilon \\ \varepsilon_1 * \cap\, \varepsilon_2 * = \varnothing \qquad x \notin \mathrm{fv}(T) \qquad y \notin \mathrm{fv}(T) \qquad \theta = [q_2/y][q_1/x]\end{array}}{\Gamma^{\varphi} \models \textbf{let } x = t_1 \textbf{ in let } y = t_2 \textbf{ in } t \approx_{\log} \textbf{let } y = t_2 \textbf{ in let } x = t_1 \textbf{ in } t : (T^{q}\ \varepsilon_1 \rhd \varepsilon_2 \rhd \varepsilon)\theta}$$

$$(\lambda\textsc{-hoist})\ \frac{\begin{array}{c}\Gamma^{\varphi} \vdash t_1 : T_1^{q_1}\ \varnothing \qquad \lceil \Gamma\,,\ x : T^{p}, y : T_1^{q_1 * \cap (\varphi, x) *} \rceil^{q, x, y} \vdash t : U^{r}\ \varepsilon \\ \theta = [q_1/y] \qquad x \notin \mathrm{fv}(U) \qquad y \notin \mathrm{fv}(U)\end{array}}{\Gamma^{\varphi} \models (\lambda x : T^{p}.\ \textbf{let } y = t_1 \textbf{ in } t) \approx_{\log} (\textbf{let } y = t_1 \textbf{ in } \lambda x : T^{p}.t) : (x : T^{p} \to^{\varepsilon} U^{r}\theta)^{q}\ \varnothing}$$

$$(\beta\textsc{-inlining})\ \frac{\lceil \Gamma, x : T^{p} \rceil^{q, x} \vdash t_1 : U^{r}\varepsilon \qquad \Gamma^{\varphi} \vdash t_2 : T^{p}\ \varnothing \qquad x \notin \mathrm{fv}(U) \qquad \theta = [p/x]}{\Gamma^{\varphi} \models (\lambda x : T^{p}.t_1)(t_2) \approx_{\log} t_1[t_2/x] : (U^{r}\ \varepsilon)\theta}$$

$$(\textsc{e-cse})\ \frac{\begin{array}{c}\Gamma^{\varphi} \vdash t_1 : T_1^{q_1}\ \varepsilon_1 \\ \lceil \Gamma\,,\ x : T_1^{q_1 * \cap \varphi *}, y : T_1^{q_1 * \cap (\varphi, x) *} \rceil^{\varphi, x, y} \vdash t : T^{q}\ \varepsilon \qquad \omega \notin \varepsilon_1 \qquad \theta = [x/y]\end{array}}{\Gamma^{\varphi} \models (\textbf{let } x = t_1 \textbf{ in let } y = t_1 \textbf{ in } t) \approx_{\log} (\textbf{let } x = t_1 \textbf{ in } t\ \theta) : (T^{q}\ \varepsilon)\theta \rhd \varepsilon_1}$$

Fig. 15. Equational rules for the λ^*_ε-calculus.

Lemma 7.41 (Congruency of Binary Logical Relations). *The binary logical relation is closed under well-typed program contexts,* i.e., *if* $\Gamma^{\varphi} \models t_1 \approx_{log} t_2 : T^{p}\ \varepsilon$, *and* $C : (\Gamma^{\varphi}; T^{p}\ \varepsilon) \Rrightarrow (\Gamma'^{\varphi'}; T'^{p'}\ \varepsilon')$, *then* $\Gamma'^{\varphi'} \models C[t_1] \approx_{log} C[t_2] : T'^{p'}\ \varepsilon'$.

Proof. By induction on the derivation of context C. Each case follows from the corresponding compatibility lemma and may use the fundamental theorem (Theorem 7.40) if necessary. □

Lemma 7.42 (Adequacy of the binary logical relations). *The binary logical relation preserves termination,* i.e., *if* $\varnothing \models t_1 \approx_{log} t_2 : T^{\varnothing}\ \varnothing$, *then* $\exists\, \sigma, \sigma', v.\ \varnothing \mid t_1 \longrightarrow_{\mathrm{v}}^{*} \sigma \mid v \wedge \varnothing \mid t_2 \longrightarrow_{\mathrm{v}}^{*} \sigma' \mid v$.

Proof. We know $(\varnothing, \varnothing) \in G[\![\varnothing]\!]$ by the interpretation of typing context. Then we can prove the result by the binary term interpretation (Fig. 14). □

Theorem 7.43 (Soundness of Binary Logical Relations). *The binary logical relation is sound w.r.t. contextually equivalence,* i.e., *if* $\Gamma^{\varphi} \vdash t_1 : T^{p}\ \varepsilon$ *and* $\Gamma^{\varphi} \vdash t_2 : T^{p}\ \varepsilon$, *then* $\Gamma^{\varphi} \models t_1 \approx_{log} t_2 : T^{p}\ \varepsilon$ *implies* $\Gamma^{\varphi} \models t_1 \approx_{ctx} t_2 : T^{p}\ \varepsilon$.

Proof. By the refined definition of contextual equivalence, to prove the result, we are given a well-typed context $C : (\Gamma^{\varphi}; T^{p}\ \varepsilon) \Rrightarrow (\varnothing; B^{\varnothing}\ \varnothing)$, and we need to show $\exists\, \sigma, \sigma', v.\ \varnothing \mid C[t_1] \longrightarrow_{\mathrm{v}}^{*} \sigma \mid v \wedge \varnothing \mid C[t_2] \longrightarrow_{\mathrm{v}}^{*} \sigma' \mid v$. By the assumption, and the congruency lemma (Lemma 7.41), we have $\varnothing \models C[t_1] \approx_{\log} C[t_2] : B^{\varnothing}\ \varnothing$, which leads to $\exists\, \sigma, \sigma', v.\ \varnothing \mid C[t_1] \longrightarrow_{\mathrm{v}}^{*} \sigma \mid v \wedge \varnothing \mid C[t_2] \longrightarrow_{\mathrm{v}}^{*} \sigma' \mid v$ by the adequacy lemma (Lemma 7.42). □

7.9 Equational Rules

Fig. 15 shows equational rules for λ^*_ε with effects specifying logically equivalent terms. Rule (dce) permits the removal of a terminating term, t_1, whose computation result is an unused value, provided that the effect of that computation only allows allocation. Removing a term with effects may not be sound, as the effects could be observed by the following computations. Rule (COMM) permits re-ordering of two terms if their effects are separate, which entails disjoint sets of store locations (Corollary 2.13). Rule (λ-HOIST) permits a pure computation to be moved out of the abstraction boundary. Rule (β-INLINING) permits replacing a function call site t_2 with the body of the called function, provided that t_2 is pure. Rule (E-CSE) permits removing a duplicated computation, provided that no fresh allocations occur during the reduction of the term. The rest of this section shows the proofs of those equational rules by using our logical relations.

LEMMA 7.44 (DEAD CODE ELIMINATION). *If $\Gamma^{\varphi} \vdash t_1 : T_1^{q_1}\ \varepsilon_1$, and $\Gamma^{\varphi} \vdash t_2 : T_2^{q_2}\ \varepsilon_2$, and $\varepsilon_1 = \varnothing$ or ω, and t_1 terminates, then $\Gamma^{\varphi} \vdash \textbf{let}\ x = t_1\ \textbf{in}\ t_2 \approx_{log} t_2 : T_2^{q_2}\varepsilon_2$.*

PROOF. By the fundamental property (Theorem 7.40), we know on the first assumption, we know $\Gamma^{\varphi} \models t_1 \approx_{\log} t_1 : T_1^{q_1}\ \varepsilon_1$.

Let $(\mathrm{W}, \gamma) \in G[[\Gamma^{\varphi}]]$ and $(\sigma_1, \sigma_2) : \mathrm{W}$. By the definition of binary logical relations and binary term interpretation, we know there exists σ_{11}, σ_{12}, W_1, v_{11} and v_{12}, such that

- $\sigma_1 \mid \gamma_1(t_1) \longrightarrow_{\mathrm{v}}^{*} \sigma_{11} \mid v_{11}$
- $\sigma_2 \mid \gamma_2(t_1) \longrightarrow_{\mathrm{v}}^{*} \sigma_{12} \mid v_{12}$
- $(\mathrm{W}; \mathrm{W}_1, v_{11}, v_{12}) \in \mathcal{V}[[\mathrm{T}_1]]^{\gamma}$
- $(\sigma_{11}, \sigma_{12}) : \mathrm{W}; \mathrm{W}_1$
- $v_{11} \rightsquigarrow^{\sigma_1} (\mathrm{locs}(\gamma_1(\varphi \cap q_1)))$
- $v_{12} \rightsquigarrow^{\sigma_2} (\mathrm{locs}(\gamma_2(\varphi \cap q_1)))$
- $\sigma_1 \hookrightarrow^{\mathrm{locs}(\gamma_1(\varepsilon_1))} \sigma_{11}$
- $\sigma_2 \hookrightarrow^{\mathrm{locs}(\gamma_2(\varepsilon_1))} \sigma_{12}$

By $\varepsilon_1 = \varnothing$ or ω, we know $\sigma_{11} = \sigma_1 * \sigma_{fr1}$, and $\sigma_{12} = \sigma_2 * \sigma_{fr2}$.
By the fundamental property (Theorem 7.40) again, we know $\Gamma^{\varphi} \models t_2 \approx_{\log} t_2 : T_2^{q_2}\ \varepsilon_2$.
By the binary term interpretation, we know there exists σ_{21}, σ_{22}, W_2, v_{21} and v_{22}, such that

- $\sigma_1 \mid \gamma_1(t_2) \longrightarrow_{\mathrm{v}}^{*} \sigma_{21} \mid v_{21}$
- $\sigma_2 \mid \gamma_2(t_2) \longrightarrow_{\mathrm{v}}^{*} \sigma_{22} \mid v_{22}$
- $(\mathrm{W}; \mathrm{W}_2, v_{21}, v_{22}) \in \mathcal{V}[[\mathrm{T}_2]]^{\gamma}$
- $(\sigma_{21}, \sigma_{22}) : \mathrm{W}; \mathrm{W}_2$
- $v_{21} \rightsquigarrow^{\sigma_1} (\mathrm{locs}(\gamma_1(\varphi \cap q_2)))$
- $v_{22} \rightsquigarrow^{\sigma_2} (\mathrm{locs}(\gamma_2(\varphi \cap q_2)))$
- $\sigma_1 \hookrightarrow^{\mathrm{locs}(\gamma_1(\varepsilon_2))} \sigma_{11}$
- $\sigma_2 \hookrightarrow^{\mathrm{locs}(\gamma_2(\varepsilon_2))} \sigma_{12}$

From the left, by the reduction semantics, we have:

- $\sigma_1 \mid \gamma_1(t_1) \longrightarrow_{\mathrm{v}}^{*} \sigma_1 * \sigma_{fr1} \mid v_{11}$
- $\sigma_1 * \sigma_{fr1} \mid \gamma_1(t_2) \longrightarrow_{\mathrm{v}}^{*} \sigma_{21} \mid v_{21}'$

By the deterministic of reduction semantics, we know $v_{21} = v_{21}'$.
Form the right, by the reduction semantics, we know $\sigma_2 \mid \gamma_2(t_2) \longrightarrow_{\mathrm{v}}^{*} \sigma_{22} \mid v_{22}$.
By the fact we have above, the proof is done.

□

LEMMA 7.45 (COMM). *If $\Gamma^{\varphi} \vdash t_1 : T_1^{q_1}\ \varepsilon_1$, and $\Gamma^{\varphi} \vdash t_2 : T_2^{q_2}\ \varepsilon_2$, and $[\Gamma\ ,\ x : T_1^{q_1 * \cap \varphi *},\ y : T_2^{q_2 * \cap (\varphi, x) *}]^{\varphi, x, y} \vdash t : T^{q}\ \varepsilon$, and $\varepsilon_1 * \cap\, \varepsilon_2 * = \varnothing$, and $x \notin \mathrm{fv}(T)$, and $y \notin \mathrm{fv}(T)$, and $\theta = [q_2/y][q_1/x]$, then $\Gamma^{\varphi} \models \textbf{let}\ x = t_1\ \textbf{in}\ \textbf{let}\ y = t_2\ \textbf{in}\ t \approx_{log} \textbf{let}\ y = t_2\ \textbf{in}\ \textbf{let}\ x = t_1\ \textbf{in}\ t : (T^{q}\ \varepsilon_1 \rhd \varepsilon_2 \rhd \varepsilon)\theta$.*

PROOF. By the fundamental property (Theorem 7.40), we know $\Gamma^{\varphi} \models t_1 \approx_{\log} t_1 : T_1^{q_1}\ \varepsilon_1$.

Let $(\mathrm{W}, \gamma) \in G[\![\Gamma^{\varphi}]\!]$ and $(\sigma_1, \sigma_2) : \mathrm{W}$. By the definition of binary logical relations and binary term interpretation, we know there exists σ_{11}, σ_{12}, W_1, v_{11} and v_{12}, such that

- $\sigma_1 \mid \gamma_1(t_1) \longrightarrow_{\mathrm{v}}^{*} \sigma_{11} \mid v_{11}$
- $\sigma_2 \mid \gamma_2(t_1) \longrightarrow_{\mathrm{v}}^{*} \sigma_{12} \mid v_{12}$
- $(\mathrm{W}; \mathrm{W}_1, v_{11}, v_{12}) \in \mathcal{V}[\![\mathrm{T}_1]\!]^{\gamma}$
- $(\sigma_{11}, \sigma_{12}) : \mathrm{W}; \mathrm{W}_1$
- $v_{11} \rightsquigarrow^{\sigma_1} (\mathrm{locs}(\gamma_1(\varphi \cap q_1)))$
- $v_{12} \rightsquigarrow^{\sigma_2} (\mathrm{locs}(\gamma_2(\varphi \cap q_1)))$
- $\sigma_1 \hookrightarrow^{\mathrm{locs}(\gamma_1(\varepsilon_1))} \sigma_{11}$
- $\sigma_2 \hookrightarrow^{\mathrm{locs}(\gamma_2(\varepsilon_1))} \sigma_{12}$

By the fundamental property (Theorem 7.40) again, we know $\Gamma^{\varphi} \models t_2 \approx_{\log} t_2 : T_2^{q_2}\ \varepsilon_2$. By the binary term interpretation, we know there exists σ_{21}, σ_{22}, W_2, v_{21} and v_{22}, such that

- $\sigma_1 \mid \gamma_1(t_2) \longrightarrow_{\mathrm{v}}^{*} \sigma_{21} \mid v_{21}$
- $\sigma_2 \mid \gamma_2(t_2) \longrightarrow_{\mathrm{v}}^{*} \sigma_{22} \mid v_{22}$
- $(\mathrm{W}; \mathrm{W}_2, v_{21}, v_{22}) \in \mathcal{V}[\![\mathrm{T}_2]\!]^{\gamma}$
- $(\sigma_{21}, \sigma_{22}) : \mathrm{W}; \mathrm{W}_2$
- $v_{21} \rightsquigarrow^{\sigma_1} (\mathrm{locs}(\gamma_1(\varphi \cap q_2)))$
- $v_{22} \rightsquigarrow^{\sigma_2} (\mathrm{locs}(\gamma_2(\varphi \cap q_2)))$
- $\sigma_1 \hookrightarrow^{\mathrm{locs}(\gamma_1(\varepsilon_2))} \sigma_{21}$
- $\sigma_2 \hookrightarrow^{\mathrm{locs}(\gamma_2(\varepsilon_2))} \sigma_{22}$

Let $\sigma_{1a} = \sigma_1 \downarrow \mathrm{locs}(\gamma_1(\varepsilon_1 *))$, and $\sigma_{1b} = \sigma_1 \downarrow \mathrm{locs}(\gamma_1(\varepsilon_2 *))$, and
$\sigma_{1c} = \sigma_1 \downarrow (\mathrm{dom}(\sigma_1) - \mathrm{locs}(\gamma_1(\varepsilon_1 *)) - \mathrm{locs}(\gamma_1(\varepsilon_2 *)))$.
By Lemma 7.24, we know,
(a) $\sigma_{11} = \sigma'_{1a} * \sigma_{1b} * \sigma_{1c} * \sigma_{fr11}$, for some σ'_{1a}, where $\sigma_{fr11} * \sigma_1$; and
(b) $\sigma_{21} = \sigma_{1a} * \sigma'_{1b} * \sigma_{1c} * \sigma_{fr21}$, for some σ'_{1b}, where $\sigma_{fr21} * \sigma_1$; and
(c) $\sigma_{fr11} * \sigma_{fr21}$.
Let $\sigma_{2a} = \sigma_2 \downarrow \mathrm{locs}(\gamma_2(\varepsilon_1 *))$, and $\sigma_{2b} = \sigma_2 \downarrow \mathrm{locs}(\gamma_2(\varepsilon_2 *))$, and
$\sigma_{2c} = \sigma_2 \downarrow (\mathrm{dom}(\sigma_2) - \mathrm{locs}(\gamma_2(\varepsilon_1 *)) - \mathrm{locs}(\gamma_2(\varepsilon_2 *)))$.
By Lemma 7.24, we know,
(d) $\sigma_{12} = \sigma'_{2a} * \sigma_{2b} * \sigma_{2c} * \sigma_{fr12}$, for some σ'_{2a}, where $\sigma_{fr12} * \sigma_2$; and
(e) $\sigma_{22} = \sigma_{2a} * \sigma'_{2b} * \sigma_{2c} * \sigma_{fr22}$, for some σ'_{2b}, where $\sigma_{fr22} * \sigma_2$; and
(f) $\sigma_{fr12} * \sigma_{fr22}$.
From the left, by the deterministic of reduction semantics and (a), we know:

- $\sigma_1 \mid \gamma_1(t_1) \longrightarrow_{\mathrm{v}}^{*} \sigma'_{1a} * \sigma_{1b} * \sigma_{1c} * \sigma_{fr11} \mid v_{11}$
- $\sigma'_{1a} * \sigma_{1b} * \sigma_{1c} * \sigma_{fr11} \mid \gamma_1(t_2) \longrightarrow_{\mathrm{v}}^{*} \sigma'_{1a} * \sigma'_{1b} * \sigma_{1c} * \sigma_{fr11} * \sigma'_{fr21} \mid v_{21}$

From the right, by the deterministic of reduction semantics and (e), we know

- $\sigma_2 \mid \gamma_2(t_2) \longrightarrow_{\mathrm{v}}^{*} \sigma_{2a} * \sigma'_{2b} * \sigma_{2c} * \sigma_{fr22} \mid v_{22}$
- $\sigma_{2a} * \sigma'_{2b} * \sigma_{2c} * \sigma_{fr22} \mid \gamma_1(t_1) \longrightarrow_{\mathrm{v}}^{*} \sigma'_{2a} * \sigma'_{2b} * \sigma_{2c} * \sigma_{fr22} * \sigma'_{fr12} \mid v_{12}$

Let $\sigma_c = \sigma'_{1a} * \sigma'_{1b} * \sigma_{1c} * \sigma_{fr11} * \sigma'_{fr21}$ and $\sigma_d = \sigma_{2a} * \sigma'_{2b} * \sigma_{2c} * \sigma_{fr22} * \sigma'_{fr12}$.
We know that there exists W', such that $(\sigma_c, \sigma_d) : \mathrm{W}; \mathrm{W}'$, and $\mathrm{W}; \mathrm{W}_1 \subseteq \mathrm{W}; \mathrm{W}'$. and $\mathrm{W}; \mathrm{W}_2 \subseteq \mathrm{W}; W'$.
By Lemma 7.7, we know that $(\mathrm{W}; \mathrm{W}', v_{11}, v_{12}) \in \mathcal{V}[\![\mathrm{T}_1]\!]^{\gamma}$, and $(\mathrm{W}; \mathrm{W}', v_{21}, v_{22}) \in \mathcal{V}[\![\mathrm{T}_2]\!]^{\gamma}$
By the reduction semantics, and binary term interpretation, we know that
As $\gamma_2(t_1)[y \mapsto v_{22}][x \mapsto v_{12}] = \gamma_2(t_1)[x \mapsto v_{12}][y \mapsto v_{22}]$, we have
$(\mathrm{W}; \mathrm{W}', \gamma_1(t_1)[x \mapsto v_{11}][y \mapsto v_{21}], \gamma_2(t_1)[y \mapsto v_{22}][x \mapsto v_{12}]) \in \mathcal{E}[\![T^{q}\ \varepsilon]\!]^{\gamma}_{\varphi}$ by the fundamental propety (Theorem 7.40) on the second assumption.

□

Lemma 7.46 (λ-hoist). *If* $\Gamma^{\varphi} \vdash t_1 : T_1^{q_1}\ \varnothing$, *and* $[\Gamma\ ,\ x : T^p, y : T_1^{q_1 * \cap (\varphi, x)*}]^{q,x,y} \vdash t : U^r\ \varepsilon$, *and* $\theta = [q_1/y]$, *and* $x \notin \mathrm{fv}(U)$, *and* $y \notin \mathrm{fv}(U)$, *then* $\Gamma^{\varphi} \models (\lambda x : T^p.\ \textbf{\textit{let}}\ y = t_1\ \textbf{\textit{in}}\ t) \approx_{log} (\textbf{\textit{let}}\ y = t_1\ \textbf{\textit{in}}\ \lambda x : T^p.t) : (x : T^p \to^{\varepsilon} U^r\theta)^q\ \varnothing$.

Proof. Let $t_\lambda \overset{\text{def}}{=} (\lambda x.\ \textbf{let}\ y = t_1\ \textbf{in}\ t)$, and $t_b \overset{\text{def}}{=} \textbf{let}\ y = t_1\ \textbf{in}\ t$.

By the fundamental property (Theorem 7.40), we know $\Gamma^{\varphi} \models t_\lambda \approx_{\log} t_\lambda : (x : T^p \to^{\varepsilon} U^r\theta)^q\ \varnothing$.

Let $(\mathrm{W}, \gamma) \in G[[\Gamma^{\varphi}]]$ and $(\sigma_1, \sigma_2) : \mathrm{W}$.

By the definition of binary logical relations and function type interpretation, we know

- $\mathrm{locs}(\gamma_1(t_\lambda)) \subseteq \mathrm{dom}_1(\mathrm{W})$
- $\mathrm{locs}(\gamma_2(t_\lambda)) \subseteq \mathrm{dom}_2(\mathrm{W})$

Let v_1, v_2, W', σ_1' and σ_2' be arbitrary, such that

- $(\sigma_1', \sigma_2') : \mathrm{W}; \mathrm{W}'$
- $(\mathrm{W}; \mathrm{W}', v_1, v_2) \in \mathcal{V}[[T]]^{\gamma}$

and we know that there exists W'', σ_1'', σ_2'', v_3, v_4, such that

(a) $\sigma_1' \mid \gamma_1(t_b)[x \mapsto v_1] \longrightarrow_{\mathrm{v}}^{*} \sigma_1'' \mid v_3$
(b) $\sigma_2' \mid \gamma_2(t_b)[x \mapsto v_2] \longrightarrow_{\mathrm{v}}^{*} \sigma_2'' \mid v_4$
(c) $(\sigma'', \sigma'') : \mathrm{W}; \mathrm{W}'; \mathrm{W}''$
(d) $(\mathrm{W}; \mathrm{W}'; \mathrm{W}'', v_3, v_4) \in \mathcal{V}[[U]]^{\gamma}$

By Theorem 7.40 again, we know $\Gamma^{\varphi} \models t_1 \approx_{\log} t_1 : T_1^{q_1}\ \varnothing$. By the definition of binary logical relations and term interpretation, we know $(\mathrm{W}, t_1, t_1) \in \mathcal{E}[[T_1^{q_1}\ \varnothing]]_{\varphi}^{\gamma}$. Then we know there exists W_1, σ_a, σ_b, v_a and v_b such that

- $\sigma_1 \mid \gamma_1(t_1) \longrightarrow_{\mathrm{v}}^{*} \sigma_a \mid vx_a$
- $\sigma_2 \mid \gamma_2(t_1) \longrightarrow_{\mathrm{v}}^{*} \sigma_b \mid vx_b$
- $(\sigma_a, \sigma_b) : \mathrm{W}; \mathrm{W}_1$
- $(\mathrm{W}; \mathrm{W}_1, vx_a, vx_b) \in \mathcal{V}[[T_1]]^{\gamma}$

$*$ $vx_a \rightsquigarrow^{\sigma_1} \mathrm{locs}(\gamma_1(\varphi)) \cap \mathrm{locs}(\gamma_1(q_1))$
$**$ $vx_b \rightsquigarrow^{\sigma_2} \mathrm{locs}(\gamma_2(\varphi)) \cap \mathrm{locs}(\gamma_2(q_1))$

By Lemma 7.20, we know $\sigma_a = \sigma_1$, $\sigma_b = \sigma_2$, and $\mathrm{W}_1 = \varnothing$. By Lemma 7.7, we know $(\mathrm{W}; \mathrm{W}', vx_a, vx_b) \in \mathcal{V}[[T_1]]^{\gamma}$.

Now, we can further specify the reduction of t_b as

(A) $\sigma_1' \mid \gamma_1(t)[y \mapsto vx_a] \longrightarrow_{\mathrm{v}}^{*} \sigma_1'' \mid v_3$
(B) $\sigma_2' \mid \gamma_2(t)[y \mapsto vx_b] \longrightarrow_{\mathrm{v}}^{*} \sigma_2'' \mid v_4$

Combining (a) and (A), and we have

(1) $\sigma_1' \mid \gamma_1(t)[x \mapsto v_1][y \mapsto vx_a] \longrightarrow_{\mathrm{v}}^{*} \sigma_1'' \mid v_3$
(2) $\sigma_2' \mid \gamma_2(t)[x \mapsto v_2][y \mapsto vx_b] \longrightarrow_{\mathrm{v}}^{*} \sigma_2'' \mid v_4$

By the second assumption and Theorem 7.40, we know

$$[\Gamma\ ,\ x : T^p, y : T_1^{q_1 * \cap (\varphi, x)*}]^{q,x,y} \models t \approx_{\log} t : U^r\ \varepsilon$$

.

By Lemma 7.11, we know $(\mathrm{W}, \gamma) \in G[[\,[\Gamma]^{q,x,y}]]$. By $*$ and $**$, we apply Lemma 7.13, and have

$$(\mathrm{W}; \mathrm{W}', \gamma; (x \mapsto (v_1, v_2)); (y \mapsto (vx_a, vx_b))) \in G[[\,[\Gamma\ ,\ x : T^p, y : T_1^{q_1 * \cap (\varphi, x)*}]^{q,x,y}]]$$

By definition of binary logic relations and function type interpretation, we

(i) $\sigma_1' \mid (\gamma_1; (x \mapsto v_1); (y \mapsto vx_a))t_1 \longrightarrow_{\mathrm{v}}^{*} \sigma_1''' \mid v_5$

(ii) $\sigma_2' \mid (\gamma_2; (x \mapsto v_2); (y \mapsto vx_a))t_1 \longrightarrow_{\mathrm{v}}^{*} \sigma_2''' \mid v_6$

We know $\gamma_1(t)[x \mapsto v_1][y \mapsto vx_a] = (\gamma_1; (x \mapsto v_1); (y \mapsto vx_a))t_1$, and $\gamma_2(t)[x \mapsto v_2][y \mapsto vx_b] = (\gamma_2; (x \mapsto v_2); (y \mapsto vx_a))t_1$.

By the deterministic of the reduction semantics, we know $v_3 = v_5$ and $v_4 = v_6$. By the fact we have, the proof is done. □

The following lemma is used in the proof of β-inlining.

Lemma 7.47. *$(\Gamma, x : T^{p})^{q,x} \vdash t_1 : U^{r}\varepsilon$, and $\Gamma^{\varphi} \vdash t_2 : T^{p}\ \varnothing$, and $x \notin \mathrm{fv}(U)$, and $\theta = [p/x]$, then $\Gamma^{\varphi} \models t_1[x/t_2] \approx_{log} t_1[x/t_2] : (U^{r}\ \varepsilon)\theta$.*

Proof. By induction on the type derivation on the first. Each case follows from the corresponding compatibility lemma. □

Lemma 7.48 (β-inlining). *If $(\Gamma, x : T^{p})^{q,x} \vdash t_1 : U^{r}\varepsilon$, and $\Gamma^{\varphi} \vdash t_2 : T^{p}\ \varnothing$, and $x \notin \mathrm{fv}(U)$, and $\theta = [p/x]$, then $\Gamma^{\varphi} \vdash (\lambda x.t_1)(t_2) \approx_{log} t_1[x/t_2] : (U^{r}\ \varepsilon)\theta$.*

Proof. By the fundamental property (Theorem 7.40), we know $\Gamma^{\varphi} \models t_2 \approx_{\log} t_2 : T^{p}\ \varnothing$.

Let $(\mathrm{W}, \gamma) \in G[[\Gamma^{\varphi}]]$ and $(\sigma_1, \sigma_2) : \mathrm{W}$. By the definition of binary logical relations and binary term interpretation, we know there exists σ_{21}, σ_{22}, W_2, v_{x1} and v_{x2}, such that

- $\sigma_1 \mid \gamma_1(t_2) \longrightarrow_{\mathrm{v}}^{*} \sigma_{21} \mid v_{x1}$
- $\sigma_2 \mid \gamma_2(t_2) \longrightarrow_{\mathrm{v}}^{*} \sigma_{22} \mid v_{x2}$
- $(\mathrm{W}; \mathrm{W}_2, v_{x1}, v_{x2}) \in \mathcal{V}[[\mathrm{T}]]^{\gamma}$
- $(\sigma_{21}, \sigma_{22}) : \mathrm{W}; \mathrm{W}_2$
- $v_{x1} \rightsquigarrow^{\sigma_1} (\mathrm{locs}(\gamma_1(\varphi)) \cap \mathrm{locs}(\gamma_1(p)))$
- $v_{x2} \rightsquigarrow^{\sigma_2} (\mathrm{locs}(\gamma_2(\varphi)) \cap \mathrm{locs}(\gamma_2(p)))$
- $\sigma_1 \hookrightarrow^{\varnothing} \sigma_{21}$
- $\sigma_2 \hookrightarrow^{\varnothing} \sigma_{22}$

By Lemma 7.20, we know $\sigma_{21} = \sigma_1$, $\sigma_{22} = \sigma_2$. Then by definition of world, $\mathrm{W}_2 = \varnothing$.

Now we know $(\mathrm{W}, v_{x1}, \gamma_2(t_2)) \in \mathcal{E}[[T^{q}\ \varnothing]]_{\varphi}^{\gamma}$ by definition of term interpretation.

We know $\Gamma^{\varphi} \models (\lambda x.t_1)(t_2) \approx_{\log} (\lambda x.t_1)(t_2) : (U^{r}\ \varepsilon)\theta$.

By the definition of binary logical relations and binary term interpretation, we know there exists σ_1', σ_2', W_1, v_{y1} and v_{y2}, such that

- $\sigma_1 \mid \gamma_1(t_1)[x \mapsto v_{x1}] \longrightarrow_{\mathrm{v}}^{*} v_{y1} \mid \sigma_1'$
- $\sigma_2 \mid \gamma_2(t_1)[x \mapsto v_{x2}] \longrightarrow_{\mathrm{v}}^{*} v_{y2} \mid \sigma_2'$
- $(\sigma_1', \sigma_2') : \mathrm{W}; \mathrm{W}_1$
- $(\mathrm{W}; \mathrm{W}_1, v_{y1}, v_{y2}) \in \mathcal{V}[[\mathrm{U}]]^{\gamma}$

By Lemma 7.47, the definition of binary logical relations and binary term interpretation, we know there exists σ_3, σ_4, W', v_3 and v_4, such that

- $\sigma_1 \mid \gamma_1(t_1[x/t_2]) \longrightarrow_{\mathrm{v}}^{*} \sigma_3 \mid v_3$
- $\sigma_2 \mid \gamma_2(t_1[x/t_2]) \longrightarrow_{\mathrm{v}}^{*} \sigma_4 \mid v_4$
- $(\sigma_3, \sigma_4) : \mathrm{W}; \mathrm{W}'$
- $(\mathrm{W}; \mathrm{W}', v_3, v_4) \in \mathcal{V}[[\mathrm{U}]]^{\gamma}$

By $(\mathrm{W}, v_{x1}, \gamma_2(t_2)) \in \mathcal{E}[[T^{q}\ \varnothing]]_{\varphi}^{\gamma}$, we know $v_4 = v_{y2}$. Then we are done.

□

The following lemma is used in the proof of Lemma 7.50.

Lemma 7.49. *If $\Gamma^{\varphi} \vdash t_1 : T_1^{q_1}\ \varepsilon_1$ and $[\Gamma\ ,\ x : T_1^{q_1 * \cap \varphi *}, y : T_1^{q_1 * \cap (\varphi, x) *}]^{\varphi, x, y} \vdash t : T^{q}\ \varepsilon$, and $\theta = [x/y]$, and $\omega \notin \varepsilon_1$, then $\Gamma^{\varphi} \models (\textbf{let}\ x = t_1\ \textbf{in}\ t\ \theta) \approx_{log} (\textbf{let}\ x = t_1\ \textbf{in}\ t\ \theta) : (T^{q}\ \varepsilon)\theta \rhd \varepsilon_1$.*

Proof. By induction on the type derivation on the second. Each case follows from the corresponding compatibility lemma. □

Lemma 7.50 (e-cse). *If* $\Gamma^{\varphi} \vdash t_1 : T_1^{q_1}\ \boldsymbol{\varepsilon_1}$, *and* $[\Gamma\,,\ x : T_1^{q_1 * \cap \varphi *}, y : T_2^{q_2 * \cap (\varphi, x) *}]^{\varphi, x, y} \vdash\ t : T^{q}\ \boldsymbol{\varepsilon}$, *and* $\omega \notin \boldsymbol{\varepsilon_1}$, *and* $\theta = [x/y]$, *then* $\Gamma^{\varphi} \models (\textbf{let}\ x = t_1\ \textbf{in let}\ y = t_1\ \textbf{in}\ t) \approx_{log} (\textbf{let}\ x = t_1\ \textbf{in}\ t\ \theta) : (T^{q}\ \boldsymbol{\varepsilon})\theta \triangleright \boldsymbol{\varepsilon_1}$.

Proof. By the fundamental property (Theorem 7.40), we know $\Gamma^{\varphi} \models t_1 \approx_{\log} t_1 : T_1^{q_1}\ \boldsymbol{\varepsilon_1}$.

Let $(\mathrm{W}, \gamma) \in G[[\Gamma^{\varphi}]]$ and $(\sigma_1, \sigma_2) : \mathrm{W}$. By the definition of binary logical relations and binary term interpretation, we know there exists σ_{11}, σ_{12}, W_1, v_{11} and v_{12}, such that

- $\sigma_1 \mid \gamma_1(t_1) \longrightarrow_{\mathrm{v}}^{*} \sigma_{11} \mid v_{11}$
- $\sigma_2 \mid \gamma_2(t_1) \longrightarrow_{\mathrm{v}}^{*} \sigma_{12} \mid v_{12}$
- $(\mathrm{W}; \mathrm{W}', v_{11}, v_{12}) \in \mathcal{V}[[\mathrm{T}_1]]^{\gamma}$
- $(\sigma_{11}, \sigma_{12}) : \mathrm{W}; \mathrm{W}_1$
- $v_{11} \rightsquigarrow^{\sigma_1} (\mathrm{locs}(\gamma_1(\varphi)) \cap \mathrm{locs}(\gamma_1(q_1)))$
- $v_{12} \rightsquigarrow^{\sigma_2} (\mathrm{locs}(\gamma_2(\varphi)) \cap \mathrm{locs}(\gamma_2(q_1)))$
- $\sigma_1 \hookrightarrow^{\mathrm{locs}(\gamma_1(\boldsymbol{\varepsilon_1}))} \sigma_{11}$
- $\sigma_2 \hookrightarrow^{\mathrm{locs}(\gamma_2(\boldsymbol{\varepsilon_1}))} \sigma_{12}$

As $\omega \notin \boldsymbol{\varepsilon_1}$, we know $\mathrm{W}_1 = \emptyset$.

From the left, by the reduction semantics, we know there exists σ_{11}, σ_{12}, W_1, v_{11} and v_{12}, such that

- $\sigma_1 \mid \gamma_1(t_1) \longrightarrow_{\mathrm{v}}^{*} \sigma_{11} \mid v_{11}$
- $\sigma_{11} \mid \gamma_1(t_1) \longrightarrow_{\mathrm{v}}^{*} \sigma_{11}' \mid v_{11}'$
- $\sigma_{11}' \mid \gamma_1(t[x \mapsto v_{11}][y \mapsto v_{11}']) \longrightarrow_{\mathrm{v}}^{*} \sigma_1' \mid v_1$

where $v_{11} = v_{11}'$ and $\sigma_{11} = \sigma_{11}'$, by the deterministic of the reduction semantics.

Thus, the last reduction step can be re-written as

$$\sigma_{11} \mid \gamma_1(t[x \mapsto v_{11}][y \mapsto v_{11}]) \longrightarrow_{\mathrm{v}}^{*} \sigma_1' \mid v_1,$$

where $\mathrm{dom}(\sigma_{11}) = \mathrm{dom}_1\, W$.

By the second assumption and Theorem 7.40, we know

$$[\Gamma\,,\ x : T_1^{q_1 * \cap \varphi *}, y : T_2^{q_2 * \cap (\varphi, x) *}]^{\varphi, x, y} \models\ t \approx_{\log} t : T^{q}\ \boldsymbol{\varepsilon}$$

.

By Lemma 7.49, we know

$$\Gamma^{\varphi} \models (\textbf{let}\ x = t_1\ \textbf{in}\ t\ \theta) \approx_{\log} (\textbf{let}\ x = t_1\ \textbf{in}\ t\ \theta) : (T^{q}\ \boldsymbol{\varepsilon})\theta \triangleright \boldsymbol{\varepsilon_1}.$$

By definition of binary logic relations and term interpretation, after reducing t_1, we know there exists σ_{1a}, σ_{2a}, W_2, v_3 and v_4, such that

- $\sigma_{11} \mid (\gamma_1)(t\theta)[x \mapsto v_{11}] \longrightarrow_{\mathrm{v}}^{*} \sigma_{1a} \mid v_3$
- $\sigma_{12} \mid (\gamma_2)(t\theta)[x \mapsto v_{12}] \longrightarrow_{\mathrm{v}}^{*} \sigma_{2a} \mid v_4$
- $(\sigma_{1a}, \sigma_{2a}) : \mathrm{W}; \mathrm{W}_1; \mathrm{W}_2$
- $v_{11} \rightsquigarrow^{\sigma_{1a}} (\mathrm{locs}(\gamma_1(\varphi)) \cap \mathrm{locs}(\gamma_1(q\ \theta)))$
- $v_{12} \rightsquigarrow^{\sigma_{2a}} (\mathrm{locs}(\gamma_2(\varphi)) \cap \mathrm{locs}(\gamma_2(q\ \theta)))$
- $\sigma_{1a} \hookrightarrow^{\mathrm{locs}(\gamma_1(\boldsymbol{\varepsilon}\theta \triangleright \boldsymbol{\varepsilon_1}))} \sigma_{11}$
- $\sigma_{2a} \hookrightarrow^{\mathrm{locs}(\gamma_2(\boldsymbol{\varepsilon}\theta \triangleright \boldsymbol{\varepsilon_2}))} \sigma_{12}$

We know $(\gamma_1(t[x/y]))[x \mapsto v_{11}] = (\gamma_1(t[v_{11}/y]))[x \mapsto v_{11}]$. Then we know $v_1 = v_3$. By the fact we have, the proof is done.

□

Context for Contextual Equivalence

$$C ::= \square \mid \textbf{let } x = C \textbf{ in } g \mid \textbf{let } x = \lambda y.C \textbf{ in } g \mid \textbf{let } x = b \textbf{ in } C$$

Context Typing Rules $\boxed{C : (\Gamma^{\varphi}; T^{q}\ \varepsilon) \Rrightarrow (\Gamma^{\varphi}; T^{q}\ \varepsilon)}$

$$\frac{\Gamma \vdash S^{p}\ \varepsilon_1 <: T^{q}\ \varepsilon_2}{\square : (\Gamma^{\varphi}; S^{p}\ \varepsilon_1) \Rrightarrow (\Gamma^{\varphi}; T^{q}\ \varepsilon_2)} \qquad (\textsc{c-hole})$$

$$\frac{\begin{array}{c} C : (\Gamma'^{\varphi'}; U'^{r'}\ \varepsilon_1) \Rrightarrow (\Gamma^{\varphi}; U^{r}\ \varepsilon_2) \quad (\Gamma,\ x : U^{r*\cap\varphi*})^{\varphi,x} \vdash_{\mathrm{M}} g : T^{p}\ \varepsilon_3 \\ x \notin \mathrm{fv}(T) \quad \theta = [r/x] \end{array}}{\textbf{let } x = C \textbf{ in } g : (\Gamma'^{\varphi'}; U'^{r'}\ \varepsilon_1) \Rrightarrow (\Gamma^{\varphi}; (T^{p}\ \varepsilon_2 \rhd \varepsilon_3)\theta)} \qquad (\textsc{c-let-1})$$

$$\frac{\begin{array}{c} C : (\Gamma'^{\varphi'}; U'^{r'}\ \varepsilon_1) \Rrightarrow ((\Gamma, y : S^{p})^{q'',y}; T^{q}\ \varepsilon_2) \\ (\Gamma, x : ((y : S^{p}) \to^{\varepsilon_2} T^{q})^{q''*\cap\varphi*})^{\varphi,x} \vdash_{\mathrm{M}} g : U^{r}\ \varepsilon_3 \\ x \notin \mathrm{fv}(U) \quad \theta = [q''/x] \quad q'' \subseteq \varphi \end{array}}{\textbf{let } x = \lambda y.C \textbf{ in } g : (\Gamma'^{\varphi'}; U'^{r'}\ \varepsilon_1) \Rrightarrow (\Gamma^{\varphi}; (U^{r}\ \varepsilon_3)\theta)} \qquad (\textsc{c-let-}\lambda)$$

$$\frac{\begin{array}{c} \Gamma^{\varphi} \vdash_{\mathrm{M}} b : S^{r}\ \varepsilon_1 \\ C : (\Gamma'^{\varphi'}; U'^{r'}\ \varepsilon_2) \Rrightarrow ((\Gamma, x : S^{r*\cap\varphi*})^{\varphi,x}; T^{p}\ \varepsilon_3) \\ x \notin \mathrm{fv}(T) \quad \theta = [r/x] \end{array}}{\textbf{let } x = b \textbf{ in } C : (\Gamma'^{\varphi'}; U'^{r'}\ \varepsilon_2) \Rrightarrow (\Gamma^{\varphi}; (T^{p}\ \varepsilon_1 \rhd \varepsilon_3)\theta)} \qquad (\textsc{c-let-2})$$

Fig. 16. Context typing rules for the λ^{*}_{M}-Calculus.

8 OPTIMIZATION RULES AND EQUATIONAL THEORY OF λ^{*}_{G}

In this section, we justify the soundness of the optimization rules shown in the main paper, *i.e.*, they equate contextually equivalent graphs. Our approach reuses the logical relation development for the direct-style $\lambda^{*}_{\varepsilon}$ system from Section 7, through the use of a "round-trip" translation exploiting the functional properties of dependency erasure and synthesis. The key point is that this is already enough: dependencies, a type-level artifact derived from the effects annotated in graphs, possess no operational meaning other than adhering to the order of runtime effects, a consequence of type soundness.

8.1 Logical and Contextual Equivalence for the Monadic Normal Form λ^{*}_{M}

As the first step, we establish that we can restrict the logical relations development for the direct-style $\lambda^{*}_{\varepsilon}$ (Section 7) to λ^{*}_{M} in MNF due to the results proved in Section 4. That is, MNF is a proper sublanguage of $\lambda^{*}_{\varepsilon}$ and preserved by reductions (Lemmas 4.1 and 4.3). Therefore, MNF is also preserved by the logical equivalence (Definition 7.3), and contextual equivalence (Definition 7.1). Note that by restricting to monadic syntax, the contexts C in Figure 12 can only be of the shape

$$C ::= \square \mid \textbf{let } x = C \textbf{ in } g \mid \textbf{let } x = \lambda y.C \textbf{ in } g \mid \textbf{let } x = b \textbf{ in } C$$

for which we can derive the specialized context typing rules (Figure 16). Finally, some equational rules on expressions (*e.g.*, (β-INLINING), Figure 15) require a translation into MNF first to fit into the syntactic constraints of λ^{*}_{M}. This is always feasible due to the totality of the translation and its type-and-effect preservation (Lemma 4.2).

Thus, without further ado, we will treat the development of Section 7 as being defined over λ^{*}_{M}.

8.2 Context Typing, Synthesis and Erasure

In this section, we establish properties about contexts used for contextual equivalence in λ^*_{G}. We essentially lift the dependency synthesis (Figure 9) to a notion of dependency synthesis for λ^*_{M} contexts (Figure 16).

Definition 8.1 (Dependency Erasure). We write $\ulcorner g \urcorner$ ($\ulcorner b \urcorner$), for the erasure of effect dependencies from λ^*_{G} graph terms (bindings), yielding their unannotated version in λ^*_{M} (Section 4), as well as $\ulcorner C \urcorner$ for erasing dependencies in λ^*_{G} contexts from Figure 17 into λ^*_{M} contexts from Figure 16.

Definition 8.2 (Context Dependency Synthesis). We write

$$\Delta \vdash C : (\Gamma'^{\varphi'}; T'^{q'}\ \boldsymbol{\varepsilon}') \Rrightarrow (\Gamma^{\varphi}; T^{q}\ \boldsymbol{\varepsilon}) \rightsquigarrow \mathbf{C} \bullet \Delta'$$

for context synthesis, obtained from lifting the dependency synthesis (Figure 9) to context typing derivations (Figure 16).

Intuitively, the synthesis over contexts yields the annotated context with respect to the ambient last-uses coeffect Δ, with Δ' being the ambient last uses at the hole of the context. The functional properties of dependency synthesis on λ^*_{M} terms (Section 6.4) carry over analogously to contexts:

Lemma 8.3 (Soundness of Context Dependency Synthesis). *If*

$$\Delta \vdash C : (\Gamma'^{\varphi'}; T'^{q'}\ \boldsymbol{\varepsilon}') \Rrightarrow (\Gamma^{\varphi}; T^{q}\ \boldsymbol{\varepsilon}) \rightsquigarrow \mathbf{C} \bullet \Delta'$$

then

$$\mathbf{C} : (\Gamma'^{\varphi'} \bullet \Delta'; T'^{q'}\ \boldsymbol{\varepsilon}') \Rrightarrow (\Gamma^{\varphi} \bullet \Delta; T^{q}\ \boldsymbol{\varepsilon})$$

Proof. By induction over the derivation $\Delta \vdash C : (\Gamma'^{\varphi'}; T'^{q'}\ \boldsymbol{\varepsilon}') \Rrightarrow (\Gamma^{\varphi}; T^{q}\ \boldsymbol{\varepsilon}) \rightsquigarrow \mathbf{C} \bullet \Delta'$. □

Lemma 8.4 (Context Dependency Synthesis is Total). *If*

$$C : (\Gamma'^{\varphi'}; T'^{q'}\ \boldsymbol{\varepsilon}') \Rrightarrow (\Gamma^{\varphi}; T^{q}\ \boldsymbol{\varepsilon})$$

then for all Δ *with* $\mathrm{dom}(\Delta) = \mathrm{dom}(\Gamma)$ *there is* Δ' *with* $\mathrm{dom}(\Delta') = \mathrm{dom}(\Gamma')$ *and an annotated context* $\mathbf{C}$ *where*

$$\Delta \vdash C : (\Gamma'^{\varphi'}; T'^{q'}\ \boldsymbol{\varepsilon}') \Rrightarrow (\Gamma^{\varphi}; T^{q}\ \boldsymbol{\varepsilon}) \rightsquigarrow \mathbf{C} \bullet \Delta'$$

and $\ulcorner \mathbf{C} \urcorner = C$.

Proof. By induction over the context typing derivation for C. □

Lemma 8.5 (Context Re-Synthesis). *If*

$$C : (\Gamma'^{\varphi'} \bullet \Delta'; T'^{q'}\ \boldsymbol{\varepsilon}') \Rrightarrow (\Gamma^{\varphi} \bullet \Delta; T^{q}\ \boldsymbol{\varepsilon})$$

then

$$\Delta \vdash \ulcorner C \urcorner : (\Gamma'^{\varphi'}; T'^{q'}\ \boldsymbol{\varepsilon}') \Rrightarrow (\Gamma^{\varphi}; T^{q}\ \boldsymbol{\varepsilon}) \rightsquigarrow C \bullet \Delta'.$$

Proof. By induction over the context typing derivation for C. □

Lemma 8.6 (Decomposition). *If* $\Gamma^{\varphi} \bullet \Delta \vdash C[\,g\,] : T^{q}\ \boldsymbol{\varepsilon}$, *then* $\Gamma'^{\varphi'} \bullet \Delta' \vdash g : S^{p}\ \boldsymbol{\varepsilon}'$ *and* $C : (\Gamma'^{\varphi'} \bullet \Delta'; S^{p}\ \boldsymbol{\varepsilon}') \Rrightarrow (\Gamma^{\varphi} \bullet \Delta; T^{q}\ \boldsymbol{\varepsilon})$ *for some* Γ', φ', Δ', S, p, *and* $\boldsymbol{\varepsilon}'$.

Proof. By induction over the context C. □

Lemma 8.7 (Plugging). *If* $\Gamma'^{\varphi'} \bullet \Delta' \vdash g : S^{p}\ \boldsymbol{\varepsilon}'$ *and* $C : (\Gamma'^{\varphi'} \bullet \Delta'; S^{p}\ \boldsymbol{\varepsilon}') \Rrightarrow (\Gamma^{\varphi} \bullet \Delta; T^{q}\ \boldsymbol{\varepsilon})$, *then* $\Gamma^{\varphi} \bullet \Delta \vdash C[\,g\,] : T^{q}\ \boldsymbol{\varepsilon}$.

Proof. By induction over the context typing for C. □

Context for Contextual Equivalence

$$C ::= \square \mid \textbf{let}\ x = C \bullet \delta\ \textbf{in}\ g \mid \textbf{let}\ x = (\lambda y.C \bullet \delta) \bullet \delta\ \textbf{in}\ g \mid \textbf{let}\ x = b \bullet \delta\ \textbf{in}\ C$$

Context Typing Rules $\boxed{C : (\Gamma^{\varphi} \bullet \Delta; T^{q}\ \varepsilon) \Rrightarrow (\Gamma^{\varphi} \bullet \Delta; T^{q}\ \varepsilon)}$

$$\frac{\Gamma \vdash S^{p}\ \varepsilon_1 <: T^{q}\ \varepsilon_2}{\square : (\Gamma^{\varphi} \bullet \Delta; S^{p}\ \varepsilon_1) \Rrightarrow (\Gamma^{\varphi} \bullet \Delta; T^{q}\ \varepsilon_2)} \quad \text{(C-HOLE)}$$

$$\frac{\begin{array}{c} C : (\Gamma'^{\varphi'} \bullet \Delta'; U'^{r'}\ \varepsilon_1) \Rrightarrow (\Gamma^{\varphi} \bullet \Delta; U^{r}\ \varepsilon_2) \quad (\Gamma, x : U^{r*\cap\varphi*})^{\varphi,x} \bullet \Delta, (\varepsilon_2*, x) \mapsto x \vdash g : T^{p}\ \varepsilon_3 \\ x \notin \mathrm{fv}(T) \quad \theta = [r/x] \quad \delta \sqsubseteq \Delta|_{\varepsilon_2*} \end{array}}{\textbf{let}\ x = C \bullet \delta\ \textbf{in}\ g : (\Gamma'^{\varphi'} \bullet \Delta'; U'^{r'}\ \varepsilon_1) \Rrightarrow (\Gamma^{\varphi} \bullet \Delta; (T^{p}\ \varepsilon_2 \rhd \varepsilon_3)\theta)} \quad \text{(C-LET-1)}$$

$$\frac{\begin{array}{c} C : (\Gamma'^{\varphi'} \bullet \Delta'; U'^{r'}\ \varepsilon_1) \Rrightarrow ((\Gamma, y : S^{p})^{q'',y} \bullet \mapsto y; T^{q}\ \varepsilon_2) \\ (\Gamma, x : ((y : S^{p}) \to^{\varepsilon_2} T^{q})^{q''*\cap\varphi*})^{\varphi,x} \bullet \Delta, (\varepsilon_4*, x \mapsto x) \vdash g : U^{r}\ \varepsilon_3 \\ x \notin \mathrm{fv}(U) \quad \theta = [q''/x] \quad q'' \subseteq \varphi \quad \delta_1 \sqsubseteq \varepsilon_2* \mapsto y \quad \delta_2 \sqsubseteq \Delta|_{\varepsilon_4*} \end{array}}{\textbf{let}\ x = (\lambda y.C \bullet \delta_1) \bullet \delta_2\ \textbf{in}\ g : (\Gamma'^{\varphi'} \bullet \Delta'; U'^{r'}\ \varepsilon_1) \Rrightarrow (\Gamma^{\varphi} \bullet \Delta; (U^{r}\ \varepsilon_3)\theta)} \quad \text{(C-LET-}\lambda\text{)}$$

$$\frac{\begin{array}{c} \Gamma^{\varphi} \bullet \Delta \vdash b : S^{r}\ \varepsilon_1 \\ C : (\Gamma'^{\varphi'} \bullet \Delta'; U'^{r'}\ \varepsilon_2) \Rrightarrow ((\Gamma, x : S^{r*\cap\varphi*})^{\varphi,x} \bullet \Delta, (\varepsilon_1*, x) \mapsto x; T^{p}\ \varepsilon_3) \\ x \notin \mathrm{fv}(T) \quad \theta = [r/x] \quad \delta \sqsubseteq \Delta|_{\varepsilon_1*} \end{array}}{\textbf{let}\ x = b \bullet \delta\ \textbf{in}\ C : (\Gamma'^{\varphi'} \bullet \Delta'; U'^{r'}\ \varepsilon_2) \Rrightarrow (\Gamma^{\varphi} \bullet \Delta; (T^{p}\ \varepsilon_1 \rhd \varepsilon_3)\theta)} \quad \text{(C-LET-2)}$$

Fig. 17. Context typing rules for the λ^*_{G} graph IR.

Lemma 8.8 (Synthesis Plugging). *If*

$$\Delta \vdash C : (\Gamma'^{\varphi'}; T'^{q'}\ \varepsilon') \Rrightarrow (\Gamma^{\varphi}; T^{q}\ \varepsilon) \rightsquigarrow \mathrm{C} \bullet \Delta'$$

and

$$\Gamma^{\varphi} \bullet \Delta' \vdash g : T'^{q'}\ \varepsilon' \rightsquigarrow \mathrm{g} \bullet \Delta'|_{\varepsilon'*}$$

then

$$\Gamma^{\varphi} \bullet \Delta \vdash C[\,g\,] : T^{q}\ \varepsilon \rightsquigarrow \mathrm{C}[\,\mathrm{g}\,] \bullet \Delta|_{\varepsilon*}.$$

Proof. By induction over the context C. □

8.3 Logical and Contextual Equivalence for λ^*_{G} with Hard Dependencies

The key point is that we can resort to the metatheory of the direct-style type-and-effect system λ^*_{ε}, because (1) MNF is a sublanguage of the direct-style language (Lemmas 4.2 and 4.3), and (2) dependencies are entirely determined by assigned effects (Lemma 6.1). Furthermore, effect dependencies have no operational meaning beyond asserting that they respect the observed call-by-value evaluation order of effects (Corollary 6.8). Thus, from those results we can appeal to the logical relation and contextual equivalence of λ^*_{M} (Section 8.1) by the erasure and re-synthesis of dependencies to derive their counterparts for λ^*_{G}:

Definition 8.9 (Logical Equivalence for λ^*_{G}).

$$\frac{\Gamma^{\varphi} \models \ulcorner g_1 \urcorner \approx_{\log} \ulcorner g_2 \urcorner : T^{q}\ \varepsilon \quad \Gamma^{\varphi} \bullet \Delta \vdash \ulcorner g_1 \urcorner : T^{q}\ \varepsilon \rightsquigarrow g_1 \bullet \Delta|_{\varepsilon*} \quad \Gamma^{\varphi} \bullet \Delta \vdash \ulcorner g_2 \urcorner : T^{q}\ \varepsilon \rightsquigarrow g_2 \bullet \Delta|_{\varepsilon*}}{\Gamma^{\varphi} \bullet \Delta \models g_1 \approx_{\log} g_2 : T^{q}\ \varepsilon}$$

Definition 8.10 (Contextual Equivalence for λ^*_{G}).

$$\frac{\Gamma^{\varphi} \models \ulcorner g_1 \urcorner \approx_{\text{ctx}} \ulcorner g_2 \urcorner : T^{q}\ \varepsilon \qquad \Gamma^{\varphi} \bullet \Delta \vdash \ulcorner g_1 \urcorner : T^{q}\ \varepsilon \leadsto g_1 \bullet \Delta|_{\varepsilon*} \qquad \Gamma^{\varphi} \bullet \Delta \vdash \ulcorner g_2 \urcorner : T^{q}\ \varepsilon \leadsto g_2 \bullet \Delta|_{\varepsilon*}}{\Gamma^{\varphi} \bullet \Delta \models g_1 \approx_{\text{ctx}} g_2 : T^{q}\ \varepsilon}$$

Intuitively, graph terms are logically/contextually equivalent iff their dependency-erased versions in λ^*_{M} are logically/contextually equivalent, and we can recover the original terms by re-synthesizing their dependencies. More precisely, they are in the image of the synthesis function with respect to the last-use coeffect Δ and the effect ε (cf. the synthesis invariant Lemma 6.1).

8.3.1 *Properties of Logical Relations.*

Theorem 8.11 (Fundamental Property). *If $\Gamma^{\varphi} \bullet \Delta \vdash g : T^{q}\ \varepsilon$, then $\Gamma^{\varphi} \bullet \Delta \models g \approx_{log} g : T^{q}\ \varepsilon$.*

Proof. $\Gamma^{\varphi} \bullet \Delta \vdash g : T^{q}\ \varepsilon$ implies $\Gamma^{\varphi} \vdash_{\text{M}} \ulcorner g \urcorner : T^{q}\ \varepsilon$ and by the fundamental Theorem 7.40, it follows that $\Gamma^{\varphi} \models \ulcorner g \urcorner \approx_{\text{log}} \ulcorner g \urcorner : T^{q}\ \varepsilon$. Finally, by Lemma 6.3, we have that $\Gamma^{\varphi} \bullet \Delta \vdash \ulcorner g \urcorner : T^{q}\ \varepsilon \leadsto g \bullet \Delta|_{\varepsilon*}$. □

Lemma 8.12 (Congruency of Binary Logical Relations). *The binary logical relation is closed under well-typed program contexts,* i.e., *if $\Gamma^{\varphi} \bullet \Delta \models g_1 \approx_{log} g_2 : T^{p}\ \varepsilon$, and $C : (\Gamma^{\varphi} \bullet \Delta; T^{p}\ \varepsilon) \Rrightarrow (\Gamma'^{\varphi'} \bullet \Delta'; T'^{p'}\ \varepsilon')$, then $\Gamma'^{\varphi'} \bullet \Delta' \models C[g_1] \approx_{log} C[g_2] : T'^{p'}\ \varepsilon'$.*

Proof.
1. By definition of logical equivalence for λ^*_{G}:
   (a) $\Gamma^{\varphi} \models \ulcorner g_1 \urcorner \approx_{\text{log}} \ulcorner g_2 \urcorner : T^{p}\ \varepsilon$.
   (b) $\Gamma^{\varphi} \bullet \Delta \vdash \ulcorner g_1 \urcorner : T^{q}\ \varepsilon \leadsto g_1 \bullet \Delta|_{\varepsilon*}$.
   (c) $\Gamma^{\varphi} \bullet \Delta \vdash \ulcorner g_2 \urcorner : T^{q}\ \varepsilon \leadsto g_2 \bullet \Delta|_{\varepsilon*}$.
2. By erasure: $\ulcorner C \urcorner : (\Gamma^{\varphi}; T^{p}\ \varepsilon) \Rrightarrow (\Gamma'^{\varphi'}; T'^{p'}\ \varepsilon')$.
3. By the congruence Lemma 8.12, and (1), (2): $\Gamma^{\varphi} \models \ulcorner C \urcorner[\ulcorner g_1 \urcorner] \approx_{\text{log}} \ulcorner C \urcorner[\ulcorner g_2 \urcorner] : T^{p}\ \varepsilon$.
4. (3) is equivalent to $\Gamma^{\varphi} \models \ulcorner C[\,g_2\,] \urcorner \approx_{\text{log}} \ulcorner C[\,g_2\,] \urcorner : T^{p}\ \varepsilon$.
5. By assumption, (2), and re-synthesis Lemma 8.5:

$$\Delta \vdash \ulcorner C \urcorner : (\Gamma^{\varphi}; T^{p}\ \varepsilon) \Rrightarrow (\Gamma'^{\varphi'}; T'^{p'}\ \varepsilon') \leadsto C \bullet \Delta'.$$

6. By (1a), (1c), (5), and synthesis plugging Lemma 8.8:
   (a) $\Gamma^{\varphi} \bullet \Delta \vdash \ulcorner C[\,g_1\,] \urcorner : T^{q}\ \varepsilon \leadsto C[\,g_1\,] \bullet \Delta|_{\varepsilon*}$.
   (b) $\Gamma^{\varphi} \bullet \Delta \vdash \ulcorner C[\,g_2\,] \urcorner : T^{q}\ \varepsilon \leadsto C[\,g_2\,] \bullet \Delta|_{\varepsilon*}$.
7. (4) and (6) prove the goal.

□

Theorem 8.13 (Soundness of Binary Logical Relations). *The binary logical relation is sound w.r.t. contextually equivalence,* i.e., *if $\Gamma^{\varphi} \bullet \Delta \vdash g_1 : T^{p}\ \varepsilon$ and $\Gamma^{\varphi} \bullet \Delta \vdash g_2 : T^{p}\ \varepsilon$, then $\Gamma^{\varphi} \bullet \Delta \models g_1 \approx_{log} g_2 : T^{p}\ \varepsilon$ implies $\Gamma^{\varphi} \bullet \Delta \models g_1 \approx_{ctx} g_2 : T^{p}\ \varepsilon$.*

Proof.
1. By definition of logical equivalence for λ^*_{G}:
   (a) $\Gamma^{\varphi} \models \ulcorner g_1 \urcorner \approx_{\text{log}} \ulcorner g_2 \urcorner : T^{p}\ \varepsilon$.
   (b) $\Gamma^{\varphi} \bullet \Delta \vdash \ulcorner g_1 \urcorner : T^{q}\ \varepsilon \leadsto g_1 \bullet \Delta|_{\varepsilon*}$.
   (c) $\Gamma^{\varphi} \bullet \Delta \vdash \ulcorner g_2 \urcorner : T^{q}\ \varepsilon \leadsto g_2 \bullet \Delta|_{\varepsilon*}$.
2. By (1a), and soundness Theorem 7.43: $\Gamma^{\varphi} \models \ulcorner g_1 \urcorner \approx_{\text{ctx}} \ulcorner g_2 \urcorner : T^{p}\ \varepsilon$.
3. The goal follows by (1b), (1c), (2), and the definition of contextual equivalence for λ^*_{G}.

□

$$\text{(DCE)}\ \frac{\begin{array}{c}\Gamma^{\varphi} \bullet \Delta \vdash b : T_1^{q_1}\ \varepsilon_1 \\ \Gamma^{\varphi} \bullet \Delta \vdash g : T_2^{q_2}\ \varepsilon_2 \qquad g \text{ terminates} \qquad \varepsilon_1 = \varnothing \text{ or } \omega \qquad \delta \sqsubseteq \Delta|_{\varepsilon_1 *}\end{array}}{\Gamma^{\varphi} \bullet \Delta \models \mathbf{let}\ x = b \bullet \delta\ \mathbf{in}\ g \approx_{\log} g : T_2^{q_2}\varepsilon_2}$$

$$\text{(COMM)}\ \frac{\begin{array}{c}\Gamma^{\varphi} \bullet \Delta \vdash b_1 : T_1^{q_1}\ \varepsilon_1 \qquad \Gamma^{\varphi} \bullet \Delta \vdash b_2 : T_2^{q_2}\ \varepsilon_2 \\ [\Gamma\ ,\ x : T_1^{q_1 * \cap \varphi *},\ y : T_2^{q_2 * \cap (\varphi, x) *}]^{\varphi, x, y} \bullet \Delta, (\varepsilon_1 *, x) \mapsto x, (\varepsilon_2 *, y) \mapsto y \vdash g : T^{q}\ \varepsilon_3 \\ \varepsilon_1 * \cap \varepsilon_2 * = \varnothing \qquad x \notin \mathrm{fv}(T) \qquad y \notin \mathrm{fv}(T) \qquad \theta = [q_2/y][q_1/x] \\ \delta_1 \sqsubseteq \Delta|_{\varepsilon_1 *} \qquad \delta_2 \sqsubseteq \Delta|_{\varepsilon_2 *}\end{array}}{\begin{array}{l}\Gamma^{\varphi} \bullet \Delta \models \quad \mathbf{let}\ x = b_1 \bullet \delta_1\ \mathbf{in}\ \mathbf{let}\ y = b_2 \bullet \delta_2\ \mathbf{in}\ g \\ \qquad\qquad \approx_{\log} \mathbf{let}\ y = b_2 \bullet \delta_2\ \mathbf{in}\ \mathbf{let}\ x = b_1 \bullet \delta_1\ \mathbf{in}\ g\ : (T^{q}\ \varepsilon_1 \triangleright \varepsilon_2 \triangleright \varepsilon_3)\theta\end{array}}$$

$$\text{(λ-HOIST)}\ \frac{\begin{array}{c}\Gamma^{\varphi} \bullet \Delta \vdash b : S^{o}\ \varnothing \qquad [\Gamma\ ,\ y : T^{p}, z : S^{o * \cap (\varphi, z) *}]^{q, y, z} \bullet \mapsto y, z \mapsto z \vdash g : U^{r}\ \varepsilon \\ \theta = [o/z] \qquad y \notin \mathrm{fv}(U) \qquad z \notin \mathrm{fv}(U)\end{array}}{\begin{array}{l}\Gamma^{\varphi} \bullet \Delta \models \mathbf{let}\ x = (\lambda y.(\mathbf{let}\ z = b \bullet \varnothing\ \mathbf{in}\ g) \bullet \delta_1) \bullet \delta_2\ \mathbf{in}\ x \\ \qquad\qquad \approx_{\log} \mathbf{let}\ z = b \bullet \varnothing\ \mathbf{in}\ \mathbf{let}\ x = (\lambda y.g \bullet \delta_1) \bullet \delta_2\ \mathbf{in}\ x \\ \qquad\qquad : ((y : T^{p}) \rightarrow^{\varepsilon} U^{r}\theta)^{q}\end{array}}$$

$$\text{(β-INLINING)}\ \frac{\begin{array}{c}[\Gamma, x : T^{p}]^{q, x} \bullet \mapsto x \vdash g : U^{r}\varepsilon \\ q \subseteq \varphi \qquad \Gamma^{\varphi} \bullet \Delta \vdash b : T^{p}\ \varnothing \qquad x \notin \mathrm{fv}(U) \qquad \theta = [p/x]\end{array}}{\begin{array}{l}\Gamma^{\varphi} \bullet \Delta \models \mathbf{let}\ x = (\lambda y.g \bullet \delta_1) \bullet \delta_2\ \mathbf{in}\ \mathbf{let}\ z = b \bullet \delta_2\ \mathbf{in}\ \mathbf{let}\ w = y\ z \bullet \delta_3\ \mathbf{in}\ w \\ \approx_{\log} \mathbf{let}\ x = (\lambda y.g \bullet \delta_1) \bullet \delta_2\ \mathbf{in}\ \mathbf{let}\ z = b \bullet \delta_2\ \mathbf{in}\ g[x \leadsto \delta_3][z/x] \qquad : (U^{r}\ \varepsilon)\theta\end{array}}$$

$$\text{(E-CSE)}\ \frac{\begin{array}{c}\Gamma^{\varphi} \bullet \Delta \vdash b : S^{p}\ \varepsilon_1 \\ [\Gamma\ ,\ x : S^{p * \cap \varphi *}, y : S^{p * \cap (\varphi, x) *}]^{\varphi, x, y} \bullet \Delta, (\varepsilon_1 *, x) \mapsto x, (\varepsilon_1 *, y) \mapsto y \vdash g : T^{q}\ \varepsilon_2 \\ \omega \notin \varepsilon_1 \qquad \theta = [x/y] \qquad \delta \sqsubseteq \Delta|_{\varepsilon_1 *}\end{array}}{\Gamma^{\varphi} \bullet \Delta \models (\mathbf{let}\ x = b \bullet \delta\ \mathbf{in}\ \mathbf{let}\ y = b \bullet \delta\ \mathbf{in}\ g) \approx_{\log} (\mathbf{let}\ x = b \bullet \delta\ \mathbf{in}\ g\theta) : T^{q}\theta\ \varepsilon_1\theta \triangleright \varepsilon_2}$$

Fig. 18. Equational rules for the λ^*_{G} graph IR. We obtain the optimization rules by congruence closure with the contexts from Figure 17.

8.4 Soundness of the Optimization Rules

From the above results about logical equivalence for the λ^*_{G} graph IR obtained by a "round-trip translation" technique, we are now equipped to prove the soundness of the main paper's optimization rules based on the results of Section 7.9. The optimization rules are the congruence closure (with respect to C contexts in Figure 17) of the equations shown in Figure 18. Those are obtainable mechanically from their counterparts in λ^*_{ε} (Figure 15) by using the type-and-effect preserving translation into MNF (Section 4.3) followed by dependency synthesis for a given map Δ of last uses (Figure 9). That is, the equational rules in Figure 18 are the dependency-annotated versions of their counterparts in MNF.

Theorem 8.14 (Compatibility of the Equational Rules). *Each rule in Figure 18 is compatible with the logical equivalence.*

Proof. Each individual rule (DCE), (COMM), (λ-HOIST), (β-INLINING), and (E-CSE) can be uniformly proved as follows:

(1) By dependency erasure, and by Lemma 4.3, we have that both graphs are equated by the direct-style version of the respective rule (Figure 15).
(2) By Lemmas 7.44 to 7.46, 7.48 and 7.50, the erased graphs are logically equivalent in λ^*_ε.
(3) By totality and soundness of synthesis (Lemmas 6.2 and 6.3), re-synthesizing the dependencies under the given Δ and context Γ^φ of the erased graphs yields the initial dependency-annotated versions.
(4) By (2) and (3), both sides are logically equivalent in λ^*_{G} (Definition 8.9).

□

Corollary 8.15 (Compatibility of the Optimization Rules). *The optimization rules for λ^*_{G},* i.e., *the congruence closure of the rules in Figure 18 is compatible in logical equivalence.*

Proof. By Theorem 8.14 and the congruency Lemma 8.12. □

Corollary 8.16 (Soundness of the Optimization Rules). *The optimization rules for λ^*_{G},* i.e., *the congruence closure of the rules in Figure 18 describe contextually equivalent graphs.*

Proof. By Corollary 8.15 and soundness Theorem 8.13. □

9 FROM GRAPHS BACK TO TREES

In this section, we discuss efficient algorithms and heuristics for code generation, which transform the graph informed by λ^*_{G} into a tree with nested structures. This section *follows the same structure as in the main paper [Bračevac et al. 2023] but explains the algorithms with greater details*: (1) we first present the basic scheduling algorithm (Section 9.1) which incorporates dead code elimination; (2) based on that, a lightweight frequency estimation heuristics (Section 9.2) introduces more flexible code motion; (3) we finally present a compact scheduling algorithm (Section 9.3) for instruction selection and inlining expressions. The optimizations are justified by the equational theory in Section 8.4.

9.1 Traversal without Redundant Code

In essence, the block scheduling algorithm traverses a hierarchy of graph-represented blocks and selects unscheduled nodes to move into tree-represented blocks (and emit code) based on the node's dependencies. Figure 19 shows the vanilla block scheduling algorithm, which operates over graph IR data structures. We use `Node` to represent a graph node, and a few auxiliary functions such as `dataDeps`/`hardDeps` to extract different kinds of dependencies of a `Node`.

At the beginning of scheduling, we have a scope of `Node`s to schedule and a symbol representing the final result of the top-level block. Transitively following the dependencies of the final result, `scheduleBlock` partitions the unscheduled nodes into two groups: (1) nodes that are scheduled in the current block, and (2) nodes that will be scheduled into other inner blocks. This process is recursively applied when encountering lambda nodes in `traverseNode`. Schedule decisions are made relying on two properties over nodes, *available* and *reachable*.

Nested Scopes. A node is *available* if all its dependent bound variables have been introduced under the current path. A node is scheduled in the `current` block if it is both reachable and available (Line 29); otherwise, it is moved to the `inner` scope (Line 33). To this end, we need to *transtively* compute the bound variables depended by nodes, of which the result is reflected by `boundDeps`. In Figure 19, `path` represents the set of the accumulated bound variables (*e.g.*, introduced by lambdas) up to the current block. Both `path` and `scope` need maintaining through the recursive calls to properly handle scopes and nested blocks.

```
 1  /* auxiliary functions to access different sorts of dependencies of a node
 2     boundDeps: dependencies that are bound variables
 3     dataDeps, effDeps: data- and effect-dependencies
 4     hardDeps (⊆ effDeps): hard effect-dependencies */
 5  val boundDeps, dataDeps, effDeps, hardDeps : Node => Set[Node]
 6  /* obtain estimated frequencies of data-/effect-dependencies of a node: */
 7  val depFreq: Node => Map[Node, Double]
 8
 9  /* traverse a single node to emit a tree node */
10  def traverseNode(inner: Set[Node], path: Set[Node], n: Node): TreeNode = n match
11    case λf(x).r =>                                          // schedule nodes into a λ scope
12      TreeNode.Scope(λf(x), scheduleBlock(inner, path ∪ {f, x}, r))
13    ...
14    case "$sym := $op($args)" =>                             // schedule common nodes as leaves
15      TreeNode.Leaf(sym, Exp(op, args))
16
17  /* schedule a block given its final result, producing a scoping tree */
18  def scheduleBlock(scope: Set[Node], path: Set[Node], res: Node): List[TreeNode] =
19    val reachable: PriorityQueue[Node] = {res}               // reachable nodes, topologically ordered
20    val reachableHard: Set[Node] = {res}                     // reachable & required by data/hard deps.
21    val reachableHot: Set[Node] = {res}                      // reachable & frequently executed
22    val current: List[Node] = ∅                              // scheduled in current block
23    val inner:   Set[Node]  = ∅                              // scheduled in inner blocks
24    def available(n: Node): Boolean = boundDeps(n) ⊆ path    // available: bound vars. in deps. are ready
25
26    for n ⟵ reachable do
27      if reachableHard(n) then                               // reachable via data/hard dependencies
28        if reachableHot(n) ∧ available(n) then               // reachable via hot paths
29          current = n :: current
30          for m ⟵ (dataDeps(n) ∪ effDeps(n)) ∩ scope do      // consider deps. hot if freq > 0.5
31            if depFreq(n)[m] > 0.5 then reachableHot += {m}
32        else                                                 // only via cold path, or hot but unavailable
33          inner += {n}
34          if reachableHot(n) then                            // deps. of unavailable hot nodes are hot
35            reachableHot += (dataDeps(n) ∪ effDeps(n)) ∩ scope
36        reachableHard += (dataDeps(n) ∪ hardDeps(n)) ∩ scope   // reach by data and only hard dependencies
37      reachable += (dataDeps(n) ∪ effDeps(n)) ∩ scope          // reach by data/effect dependencies
38
39    for n ⟵ current yield traverseNode(inner, path, n)       // recursively build up the scoping tree
```

Fig. 19. The pseudocode of the basic scheduling algorithm with two extensions. Function `scheduleBlock` decides which nodes are scheduled into the `current` scope and recursively schedules `inner` scopes. To generate code for a graph `g`, we make the call `scheduleBlock(g.nodes, ∅, g.result)`. The extension to eliminate dead code by soft dependencies (cf. Section 9.1) is marked in pink, and the extension for frequency estimation and code motion (cf. Section 9.2) in teal.

Dead Code Elimination. A node is *reachable* if it can be back-tracked from the current result node through effect or data dependencies. Only reachable nodes are considered for scheduling, which naturally eliminates dead code (cf. rule DCE, Figure 18). In Figure 19, `reachable` is a priority queue which reflects the property and enforces the topological ordering. It is populated with data and effect dependencies along the iteration (Line 37). As an extension, we discern soft dependencies (Section 6) and identify data and hard dependencies as `reachableHard` (Line 36). This ensures nodes that are only reachable via soft dependencies can be eliminated.

Complexity. Given the total number of nodes n and the maximal depth of nested blocks k, the worst-case asymptotic time complexity is bound by $O(kn^2)$. This is because the algorithm traverses over the reachable nodes in order (bound by n, $O(n)$ each), and repeats this process for nested scopes. In practice, the complexity is bound by $O(kn \log n)$ given the decreasing size of nested scopes and the limited degrees of graph nodes. To exemplify, scheduling of symbolic execution (cf. [Bračevac et al. 2023], Section 7.2) takes 19.3 sec for 548,976 graph nodes, which is rather efficient.

9.2 Code Motion by Frequency Estimation

Our basic scheduling algorithm eagerly schedules nodes to their outermost block, following the equational rules (COMM) and (λ-HOIST) in Figure 18. This is a form of code motion for no extra effort and generally desirable for functions and loops. For instance,

```
List(1, 2, 3, 4, 5).map(x => x * factorial(N))
```

Lifting the expensive `factorial` out of the lambda is beneficial and feasible since it does not depend on the bound variable `x`. However, this does not always generate optimal code. Consider a conditional expression that transforms an array of complex numbers only in the then-branch,

```
if (cnd) compNums.map(f) else compNums
```

Since `compNums.map(f)` has no dependency on `cnd`, this statement would be lifted to the outer scope and always executed regardless of the condition, thus imposing unnecessary runtime overhead when the else-branch is actually taken.

To avoid this situation, we can *estimate* how frequently a node is used and move less frequent nodes (*i.e.*, cold) into inner scopes. We assign a number to each node based on its dependents which represents how relatively often the node is executed at runtime. The results of functions and loops are assigned 100, indicating that they and their dependencies can be executed multiple times (definitely hot). The results of conditional branches are assigned 0.5 (cold), assuming each branch is taken with equal probability. All other nodes are assigned 1.0 (normal). Numbers above are illustrative and context-insensitive. Alternative metrics are possible, while what we present here is beneficial to many code patterns.

Figure 19 highlights in teal the changes to the basic scheduling algorithm to use frequency estimation. Given a node ready to schedule in the current scope (Line 28-31), we use the function `depFreq` to access the frequency estimation of its dependencies. Only those with frequencies greater than 0.5 are considered hot-reachable, and thus can be included in `current`. Others are classified to be cold, and are scheduled in inner blocks if all reaching dependencies are cold. Given a node scheduled in inner blocks (Line 32-35), its dependencies are considered to have the same level of warmth as the node itself, ensuring consistent code motion behavior for code with nested scopes.

The proposed heuristic works as expected in that it (1) lifts computation out of hot constructs such as loops, and (2) sinks computation into cold constructs such as conditionals. Regarding nested scopes, the heuristic prioritizes (1) over (2). Suppose there is a loop in the current scope and a conditional inside that loop. For a node inside the conditional, the heuristic tends to lift it out to the current scope, as the result is still used multiple times during the loop. On the other hand, if there is a loop inside a conditional in the current scope, a node used in the loop is not lifted to the current scope, but still subject to lifting up to the conditional scope to optimize the loop.

Our approach is simpler to implement and more efficient compared to more sophisticated analyses, such as lazy code motion [Knoop et al. 1992], partial redundancy elimination [Kennedy et al. 1999], or even whole-program dataflow analyses. The estimation heuristics associates a constant factor to each node traversed. Therefore, it does not change the complexity of the basic scheduling algorithm.

9.3 Instruction Selection with Compact Traversal

The basic scheduling algorithm generates code that binds every intermediate expression using `let`. The result, nevertheless, is not only verbose but also suboptimal, without using the target-specific primitives. Consider the following tensor computation snippet,

```
val X = Matmul(A, B); val C = Add(C, X); C
```

The unique use of `X` in `Add` enables further transformation into destination-passing style using generalized matrix multiplication `GEMM`, which updates `C` in-place by $C \longleftarrow \alpha AB + \beta C$. Thus, we can

```
/* shouldInline is a mutable set of nodes that are (1) locally defined,
   (2) locally used as value exactly once, and (3) have no inner use. */
val shouldInline: Set[Node] = { n ∈ localDef | currentValUse[n] == 1 ∧ innerValUse[n] == 0 }

/* seen is the set of processed nodes (cf. processNodeHere) */
val seen: Set[Node] = ∅

/* check if all successors of node n have been processed; exclude it if not. */
def checkInline(n: Node): Unit =
  if shouldInline(n) ∧ ∀ s ∈ succ[n]. seen(s) then
    processNodeHere(n)
  else shouldInline -= {n}

/* considering node n, try to inline all its dependencies */
def processNodeHere(n: Node): Unit =
  seen += {n}
  for s ← dataDeps(n).reverse do                      // respects order of argument evaluation
    checkInline(s)

/* traverse and emit a single node */
def traverseNode(inner: Set[Node], path: Set[Node], inlined: Set[Node], n: Node): TreeNode =
  def traverseInlined(n: Node): Exp = n match
    case "$sym := $op($args)" =>                      // recursively handling inlined expressions
      Exp(op, for m ← args yield if inlined(m) then traverseInlined(m) else m)
    ...

  n match
    case λf(x).r =>                                   // schedule nodes into a λ scope
      TreeNode.Scope(λf(x), scheduleBlock(inner, path ∪ {f, x}, r))
    ...
    case "$sym := $op($args)" =>                      // schedule common nodes as leaves
      TreeNode.Leaf(sym, traverseInlined(n))
```

Fig. 20. The auxiliary functions for the block scheduling algorithm with compact traversal. `shouldInline`, `seen`, `checkInline`, and `processNodeHere` are defined within `scheduleBlock` in Figure 21, while `traverseNode` replaces its former version in Figure 19. Usages of inlining information are marked in yellow.

match the tree structure `Add(C, Matmul(A, B))` and generate a single operation, eliminating the intermediate multiplication `X`:

```
GEMM(A, B, C, alpha=1.0, beta=1.0); C
```

This is basically a form of *instruction selection* seen in optimizing compilers. However, the procedure can be non-trivial on computation graphs where all consumers of a value need accounting for. A proper solution on graphs like LLVM's `SelectionDAG` [Lattner and Adve 2004] takes effort to compose and time to execute.

For the λ^*_{G} graph IR, we perform a simple but highly useful alternative called *compact traversal*: we first turn the graph nodes into inlined trees whenever possible, and then use tree matching algorithms (*e.g.*, maximal munch) to select the best primitive. Compact traversal must respect the dependencies to preserve the semantics. Consider the following code that reads the value from cell `x` and then increments the value of `x`:

```
if (cond) { val y = !x; inc(x); println(y) }
```

Since `inc(x)` depends on node `y` in an effectful way, inlining `y` to `println` breaks the semantics.

Compact traversal works on each `current` scope determined by the basic scheduling algorithm (cf. Figure 19). Initially, all nodes in the scope are viewed as individual trees. Figure 21 shows the updated main function `scheduleBlock` with compact traversal atop other extensions, and Figure 20 shows accompanying auxiliary definitions. In the following, we summarize the compact traversal algorithm in three steps.

(1) **Track Node Usage.** The key idea of compact traversal is that we only consider inlining for nodes that are locally defined and locally used exactly once, and are not used in nested scopes. To this end, we first identify all candidate nodes for inlining by tracking node usage. When scheduling

```
 1  def scheduleBlock(scope: Set[Node], path: Set[Node], res: Node): List[TreeNode] =
 2    ...                                                    // see Figure 19 for initial definitions
 3
 4    /* (1) a backward pass to track node usages, integrated with the basic scheduling algorithm */
 5    val localDef: Set[Node] = ∅                            // the collection of local definitions
 6    val innerValUse: Map[Node, Int] = {_ ↦ 0}              // how often a symbol is used in inner blocks
 7    val currentValUse: Map[Node, Int] = {res ↦ 1, _ ↦ 0}   // how often a symbol is used in current block
 8    for n ⟵ reachable do
 9      if reachableHard(n) then
10        if reachableHot(n) ∧ available(n) then
11          current = n :: current
12          localDef += {n}                                  // record local definition
13          for m ⟵ dataDeps(n) ∩ scope do                   // tracking node usage
14            if depFreq(n)[m] = 1.0 then currentValUse[m]++
15            else innerValUse[m]++
16          for m ⟵ (dataDeps(n) ∪ effDeps(n)) ∩ scope do
17            if depFreq(n)[m] > 0.5 then reachableHot += {m}
18        else
19          inner += {n}
20          for m ⟵ dataDeps(n) ∩ scope do                   // tracking node usage
21            innerValUse[m]++
22          if reachableHot(n) then
23            reachableHot += (dataDeps(n) ∪ effDeps(n)) ∩ scope
24        reachableHard += (dataDeps(n) ∪ hardDeps(n)) ∩ scope
25      reachable += (dataDeps(n) ∪ effDeps(n)) ∩ scope
26
27    /* (2) a forward pass to compute local successors */
28    val succ: Map[Node, Set[Node]] = {_ ↦ ∅}
29    for c ⟵ current do
30      for m ⟵ (dataDeps(c) ∪ effDeps(c)) ∩ localDef do
31        succ[m] += {c}
32
33    /* (3) a backward pass to check if all successors are emitted after the point of inlining */
34    val shouldInline: Set[Node] = ...                      // see Figure 20 for definitions
35    def checkInline(n: Node): Unit = ...
36    def processNodeHere(n: Node): Unit = ...
37    checkInline(res)                                       // inline the result node into "return" statement
38    for n ⟵ current.reverse do                             // process all possible inline locations
39      if !shouldInline(n) then processNodeHere(n)
40
41    /* (finally) a forward pass to perform the acutal code emission */
42    for n ⟵ current if !shouldInline(n)                    // emit each non-inlinable node
43      yield traverseNode(inner, path, shouldInline, n)     // with inlining information (cf. Figure 20)
```

Fig. 21. The pseudocode of block scheduling algorithm with compact traversal. Changes for elimination by soft dependencies are highlighted in pink, frequency estimation in teal, and compact traversal in yellow. Auxiliary function definitions can be found in Figure 20.

code, we keep track of local definitions and their symbolic names (Line 12). We also track how many times a locally defined node is used in the current scope and inner scopes as a proper value (Line 13-15, 20-21), respectively.

(2) **Compute Local Successors.** After recording the node usage information in the first step, we need to calculate local successors of a node in the current block (Line 28-31). A node x is a local successor of node y if x and y are scheduled in the same block and x depends on y. The algorithm uses a map to store the local successors of a locally defined node.

(3) **Check Inlining.** Lastly, the algorithm runs a backward pass for all nodes that can be inlined. This pass checks if all successors of a node are emitted after the point considered for inlining (Figure 20, Line 9-12). If not, we disable its inlining. Otherwise, we inline this node and check if any other nodes used by this node can be further inlined (Figure 20, Line 15-18).

A Flexible Framework for Optimization Opportunities. In Section 9 we have illustrated a series of optimizations possible by simple and composable algorithms through scheduling graph IR back to nested tree representations. Opportunities are not limited to the ones aforementioned. For instance, the basic scheduling algorithm can introduce *instruction scheduling* by assigning a proper priority value to each node reflecting not only dependency but also timing, thus tweaking the behavior of the traversal. To conclude, graph IR can be an efficient and flexible system to generate performant code.

ACKNOWLEDGMENTS

A large number of people have contributed to the design of the LMS graph IR over the years [Amin and Rompf 2017; Brown et al. 2016, 2011; Essertel et al. 2018, 2021; George et al. 2014; Lee et al. 2011; Moldovan et al. 2019; Ofenbeck et al. 2017; Rompf 2012, 2016; Rompf et al. 2015; Rompf and Odersky 2012; Rompf et al. 2013, 2011; Stojanov et al. 2019; Sujeeth et al. 2014, 2013; Tahboub et al. 2018; Wang et al. 2019a,b; Wei et al. 2023b], and in particular to the system of effect dependencies: Nada Amin, Thaïs Baudon, Kevin J. Brown, James M. Decker, Grégory Essertel, Georg Ofenbeck, Alen Stojanov, Arvind K. Sujeeth, Fei Wang, and Yushuo Xiao. This work was supported in part by NSF awards 1553471, 1564207, 1918483, 1910216, DOE award DE-SC0018050, as well as gifts from Meta, Google, Microsoft, and VMware.

REFERENCES

Amal Ahmed, Derek Dreyer, and Andreas Rossberg. 2009. State-dependent representation independence. In *POPL*. ACM, 340–353.

Amal Jamil Ahmed. 2004. *Semantics of types for mutable state.* Princeton University.

Nada Amin and Tiark Rompf. 2017. LMS-Verify: abstraction without regret for verified systems programming. In *POPL*. ACM, 859–873.

Andrew W. Appel and David A. McAllester. 2001. An indexed model of recursive types for foundational proof-carrying code. *ACM Trans. Program. Lang. Syst.* 23, 5 (2001), 657–683.

Anindya Banerjee, David A. Naumann, and Stan Rosenberg. 2013. Local Reasoning for Global Invariants, Part I: Region Logic. *J. ACM* 60, 3 (2013), 18:1–18:56.

Yuyan Bao, Gary T. Leavens, and Gidon Ernst. 2015. Conditional effects in fine-grained region logic. In *FTfJP*. ACM, 5:1–5:6.

Yuyan Bao, Gary T. Leavens, and Gidon Ernst. 2018. Unifying separation logic and region logic to allow interoperability. *Formal Aspects Comput.* 30, 3-4 (2018), 381–441.

Yuyan Bao, Guannan Wei, Oliver Bračevac, Yuxuan Jiang, Qiyang He, and Tiark Rompf. 2021. Reachability types: tracking aliasing and separation in higher-order functional programs. *Proc. ACM Program. Lang.* 5, OOPSLA (2021), 1–32.

Yuyan Bao, Guannan Wei, Oliver Bračevac, and Tiark Rompf. 2023. Modeling Reachability Types with Logical Relations: Semantic Type Soundness, Termination, and Equational Theory. arXiv:2309.05885 [cs.PL]

Nick Benton, Andrew Kennedy, Lennart Beringer, and Martin Hofmann. 2007. Relational semantics for effect-based program transformations with dynamic allocation. In *PPDP*. ACM, 87–96.

Alexander Borgida, John Mylopoulos, and Raymond Reiter. 1995. On the Frame Problem in Procedure Specifications. *IEEE Trans. Software Eng.* 21, 10 (1995), 785–798.

Oliver Bračevac, Guannan Wei, Songlin Jia, Supun Abeysinghe, Yuxuan Jiang, Yuyan Bao, and Tiark Rompf. 2023. Graph IRs for Impure Higher-Order Languages – Making Aggressive Optimizations Affordable with Precise Effect Dependencies. *Proc. ACM Program. Lang.* 7, OOPSLA2 (2023), 236:1–236:30.

Kevin J. Brown, HyoukJoong Lee, Tiark Rompf, Arvind K. Sujeeth, Christopher De Sa, Christopher R. Aberger, and Kunle Olukotun. 2016. Have abstraction and eat performance, too: optimized heterogeneous computing with parallel patterns. In *CGO*. ACM, 194–205.

Kevin J. Brown, Arvind K. Sujeeth, HyoukJoong Lee, Tiark Rompf, Hassan Chafi, Martin Odersky, and Kunle Olukotun. 2011. A Heterogeneous Parallel Framework for Domain-Specific Languages. In *PACT*. IEEE Computer Society, 89–100.

Grégory M. Essertel, Ruby Y. Tahboub, James M. Decker, Kevin J. Brown, Kunle Olukotun, and Tiark Rompf. 2018. Flare: Optimizing Apache Spark with Native Compilation for Scale-Up Architectures and Medium-Size Data. In *OSDI*. USENIX Association, 799–815.

Grégory M. Essertel, Ruby Y. Tahboub, and Tiark Rompf. 2021. On-stack replacement for program generators and source-to-source compilers. In *GPCE*. ACM, 156–169.

Cormac Flanagan, Amr Sabry, Bruce F. Duba, and Matthias Felleisen. 1993. The Essence of Compiling with Continuations. In *Proceedings of the ACM SIGPLAN'93 Conference on Programming Language Design and Implementation (PLDI), Albuquerque, New Mexico, USA, June 23-25, 1993*, Robert Cartwright (Ed.). ACM, 237–247.

Nithin George, HyoukJoong Lee, David Novo, Tiark Rompf, Kevin J. Brown, Arvind K. Sujeeth, Martin Odersky, Kunle Olukotun, and Paolo Ienne. 2014. Hardware system synthesis from Domain-Specific Languages. In *FPL*. IEEE, 1–8.

Colin S. Gordon. 2021. Polymorphic Iterable Sequential Effect Systems. *ACM Trans. Program. Lang. Syst.* 43, 1 (2021), 4:1–4:79.

John Hatcliff and Olivier Danvy. 1994. A Generic Account of Continuation-Passing Styles. In *POPL*. ACM Press, 458–471.

Robert Kennedy, Sun Chan, Shin-Ming Liu, Raymond Lo, Peng Tu, and Fred Chow. 1999. Partial Redundancy Elimination in SSA Form. *ACM Trans. Program. Lang. Syst.* 21, 3 (may 1999), 627–676.

Jens Knoop, Oliver Rüthing, and Bernhard Steffen. 1992. Lazy Code Motion. In *Proceedings of the ACM SIGPLAN 1992 Conference on Programming Language Design and Implementation* (San Francisco, California, USA) *(PLDI '92)*. Association for Computing Machinery, New York, NY, USA, 224–234.

Chris Lattner and Vikram S. Adve. 2004. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *2nd IEEE / ACM International Symposium on Code Generation and Optimization (CGO 2004), 20-24 March 2004, San Jose, CA, USA*. IEEE Computer Society, 75–88.

HyoukJoong Lee, Kevin J. Brown, Arvind K. Sujeeth, Hassan Chafi, Tiark Rompf, Martin Odersky, and Kunle Olukotun. 2011. Implementing Domain-Specific Languages for Heterogeneous Parallel Computing. *IEEE Micro* 31, 5 (2011), 42–53.

K. Rustan M. Leino. 2010. Dafny: An Automatic Program Verifier for Functional Correctness. In *LPAR (Dakar) (Lecture Notes in Computer Science, Vol. 6355)*. Springer, 348–370.

Dan Moldovan, James M. Decker, Fei Wang, Andrew A. Johnson, Brian K. Lee, Zachary Nado, D. Sculley, Tiark Rompf, and Alexander B. Wiltschko. 2019. AutoGraph: Imperative-style Coding with Graph-based Performance. In *MLSys*. mlsys.org.

Georg Ofenbeck, Tiark Rompf, and Markus Püschel. 2017. Staging for generic programming in space and time. In *GPCE*. ACM, 15–28.

Benjamin C. Pierce. 2004. *Advanced Topics in Types and Programming Languages*. The MIT Press.

John C. Reynolds. 2002. Separation Logic: A Logic for Shared Mutable Data Structures. In *LICS*. IEEE Computer Society, 55–74.

Tiark Rompf. 2012. *Lightweight Modular Staging and Embedded Compilers - Abstraction without Regret for High-Level High-Performance Programming*. Ph.D. Dissertation. EPFL, Switzerland.

Tiark Rompf. 2016. Reflections on LMS: exploring front-end alternatives. In *SCALA*. ACM, 41–50.

Tiark Rompf, Kevin J. Brown, HyoukJoong Lee, Arvind K. Sujeeth, Manohar Jonnalagedda, Nada Amin, Georg Ofenbeck, Alen Stojanov, Yannis Klonatos, Mohammad Dashti, Christoph Koch, Markus Püschel, and Kunle Olukotun. 2015. Go Meta! A Case for Generative Programming and DSLs in Performance Critical Systems. In *SNAPL (LIPIcs, Vol. 32)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 238–261.

Tiark Rompf and Martin Odersky. 2012. Lightweight modular staging: a pragmatic approach to runtime code generation and compiled DSLs. *Commun. ACM* 55, 6 (2012), 121–130.

Tiark Rompf, Arvind K. Sujeeth, Nada Amin, Kevin J. Brown, Vojin Jovanovic, HyoukJoong Lee, Manohar Jonnalagedda, Kunle Olukotun, and Martin Odersky. 2013. Optimizing data structures in high-level programs: new directions for extensible compilers based on staging. In *POPL*. ACM, 497–510.

Tiark Rompf, Arvind K. Sujeeth, HyoukJoong Lee, Kevin J. Brown, Hassan Chafi, Martin Odersky, and Kunle Olukotun. 2011. Building-Blocks for Performance Oriented DSLs. In *DSL (EPTCS, Vol. 66)*. 93–117.

Alen Stojanov, Tiark Rompf, and Markus Püschel. 2019. A stage-polymorphic IR for compiling MATLAB-style dynamic tensor expressions. In *GPCE*. ACM, 34–47.

Arvind K. Sujeeth, Kevin J. Brown, HyoukJoong Lee, Tiark Rompf, Hassan Chafi, Martin Odersky, and Kunle Olukotun. 2014. Delite: A Compiler Architecture for Performance-Oriented Embedded Domain-Specific Languages. *ACM Trans. Embed. Comput. Syst.* 13, 4s (2014), 134:1–134:25.

Arvind K. Sujeeth, Tiark Rompf, Kevin J. Brown, HyoukJoong Lee, Hassan Chafi, Victoria Popic, Michael Wu, Aleksandar Prokopec, Vojin Jovanovic, Martin Odersky, and Kunle Olukotun. 2013. Composition and Reuse with Compiled Domain-Specific Languages. In *ECOOP (Lecture Notes in Computer Science, Vol. 7920)*. Springer, 52–78.

Ruby Y. Tahboub, Grégory M. Essertel, and Tiark Rompf. 2018. How to Architect a Query Compiler, Revisited. In *SIGMOD Conference*. ACM, 307–322.

Jacob Thamsborg and Lars Birkedal. 2011. A kripke logical relation for effect-based program transformations. In *ICFP*. ACM, 445–456.

Amin Timany, Robbert Krebbers, Derek Dreyer, and Lars Birkedal. 2022. A Logical Approach to Type Soundness. https://iris-project.org/pdfs/2022-submitted-logical-type-soundness.pdf.

Fei Wang, Guoyang Chen, Weifeng Zhang, and Tiark Rompf. 2019a. Parallel Training via Computation Graph Transformation. In *IEEE BigData*. IEEE, 3430–3439.

Fei Wang, Daniel Zheng, James M. Decker, Xilun Wu, Grégory M. Essertel, and Tiark Rompf. 2019b. Demystifying differentiable programming: shift/reset the penultimate backpropagator. *Proc. ACM Program. Lang.* 3, ICFP (2019), 96:1–96:31.

Guannan Wei, Oliver Bračevac, Songlin Jia, Yuyan Bao, and Tiark Rompf. 2023a. Polymorphic Reachability Types: Tracking Freshness, Aliasing, and Separation in Higher-Order Generic Programs. arXiv:2307.13844 [cs.PL]

Guannan Wei, Songlin Jia, Ruiqi Gao, Haotian Deng, Shangyin Tan, Oliver Bracevac, and Tiark Rompf. 2023b. Compiling Parallel Symbolic Execution with Continuations. In *ICSE*. IEEE, 1316–1328.