Resource Aware Program Synthesis

Resource Analysis for Synquid

Benjamin Lichtman

Advised by Jan Hoffmann

A thesis submitted in partial fulfillment of the requirements for the Honors Undergraduate Research Thesis Program

Computer Science Department

School of Computer Science

Carnegie Mellon University

May 5, 2017

Abstract

Researchers have developed several approaches for automatically synthesizing a program from a specification. One such approach uses refinement types, which associate program types with boolean formulae that constrain the permissible inhabitants of the type. These are so expressive that they can be used as specifications for type-directed program synthesis for many recursive programs. This particular model is advantageous, as it allows the system to reject ill-typed programs during synthesis. In addition, refinement types also provide a verifiable proof of correctness for the resulting program. However, this method has no control over the resource usage of the the programs it generates. As a result, it commonly produces inefficient programs. To aid this issue, this thesis enriches the refinement-type system with the ability to calculate resource bounds. To do so, we introduce an affine-typed foundation for type-directed program synthesis that allows for automatic amortized resource analysis, or AARA. Due to the integration with AARA, this type system is able to efficiently and accurately perform static prediction of resource usage for a wide range of functional programs. By introducing linear dependent types to our system and showing its soundness with respect to a new big-step operational semantics, our approach preserves the strengths of previous methods for resource analysis and type-directed program synthesis. Our combination of these two techniques thus enables the refinement type specification to not only define the functional behavior of the desired program but its resource consumption as well.

Acknowledgements

First and foremost, I would like to thank my research advisor, Jan Hoffmann. I have thoroughly enjoyed working with you this past year and a half. Over this time, I feel like I have learned so much about being a student, a researcher, and a person. I am so grateful for all the time and effort you have given to me.

Additionally, I would like to thank Nadia Polikarpova for her support over this past year. Even though this has probably been one of the busiest years of your life, you were always able to help me through the most difficult parts of the work. Without you, this project simply could not have happened.

To Tom Cortina, my academic advisor, thank you for your guidance over the years. You have consistently pushed me to be a better student beyond what I could have ever imagined for myself. You have seen me at my proudest and at my most panicked and handled both with grace and compassion.

To Colin Angevine, my first computer science teacher, thank you for continually inspiring me to achieve success while staying humble. You alone showed me how to capture my love of language and combine it with my passion for mathematics, thereby laying the foundation that brought me to this point.

I owe so much to my friends and their support. Specifically, I would like to thank Jordan Brown and William Howard-Matchen for proofreading this thesis, the brothers of Alpha Epsilon Pi for always supporting me in every aspect of my life, and Aqeel Phillips for being my thesis buddy during this process. Most of all, I would like to thank Alex Frieder for being my main partner-in-crime since Homework 2 of 15-151. You have been the cornerstone of my entire experience at CMU.

To my mother, Anne Robbins, thank you for always being there for me in every way. I am so lucky to call you my mother, and I hope I have made you proud. To my father, Craig Lichtman, thank you for pushing me to be the best version of myself. Without your support, coaching, and extremely high expectations, I would not be who I am today.

Last but certainly not least, I would like to thank Emma Gossard. You have been my rock since day one of this process (and before) and have stuck with me through the best and worst of this past year. You fill my life with such happiness and joy, and I cannot imagine having done this without you by my side.

Contents

List of Figures

Chapter 1

Introduction

In the field of program synthesis, there are currently many approaches for specifying programs to be synthesized, including pre- and post-conditions [15], input/output examples [7], and refinement types [20]. While some of these approaches are easy to verify but only weakly expressive, others can provide strong guarantees but difficult to verify. However, Polikarpova, et. al [20] have created a synthesis technique, titled Synquid, that aims to be both expressive and easily verifiable.

Synquid, which stands for "Synthesis with Liquid Types", uses a particular family of polymorphic refinement types called *liquid types* as program specifications and develops recursive programs from those types through a novel type checking process [20]. From the provided specification, the Synquid type checking approach, titled *local liquid type checking*, propagates type information top-down in order to break the initial specification into parts, then, upon finding a term that cannot be broken down any more, infers the type of that term in a bottom-up manner. Synquid is able to synthesize implementations of a wide range of common recursive programs, including list and tree operations such as `map` and `fold`, as well as sorting algorithms and operations on structures such as binary search trees, AVL trees, and red-black trees.

One particularly representative example of the Synquid system is the `compress` function. To begin to represent a specification for this function, we need a data type that represents an ordered list as follows:

```
data PList α <p::x:α → y:PList α → Bool> where
  Nil::PList α <p>
  Cons::x:α → xs:{Plist α | p x ν} <p> → PList α <p>
```

Here, we have a datatype `PList` parameterized by a binary relation `p` that must hold between the head and tail of every `Cons`. Using this datatype, we can now define two types to be used in the specification for

`compress`, where `heads` maps a `PList` to the set containing its head element:

$$\texttt{type List } \alpha \ = \texttt{ PList } \alpha \texttt{ <}\top\texttt{>}$$

$$\texttt{type CList } \alpha \ = \texttt{ PList } \alpha \texttt{ <}\neg(\texttt{x} \in \texttt{heads y})\texttt{>}$$

With these types, we can now produce the following specification for `compress`:

$$\texttt{compress :: xs: List } \alpha \rightarrow \{\texttt{CList } \alpha \texttt{ | elems } \nu \texttt{ == elems xs}\}$$

As written, this type specification requires the `compress` function to take in an arbitrary `List` (where there is no restriction on the `Cons` operation) and return a list containing the same elements as the input where no two consecutive elements are equal. When the Synquid system is given this specification, it returns the following implementation of `compress`.

```
compress = λxs.  match xs with
  Nil → Nil
  Cons x3 x4 →
    match compress x4 with
      Nil → Cons x3 Nil
      Cons x10 x11 →
        if x3 == x10
          then compress x4
          else Cons x3 (Cons x10 x11)
```

Many program synthesis techniques, including Synquid, are prone to producing implementations of a specification that are inefficient in cases where more efficient versions exist. For example, in the above case, the implementation of `compress` that is produced runs in exponential time, whereas simply replacing the second recursive call with `Cons x10 x11` would result in a linear solution. This phenomenon is due to the fact that most approaches to synthesis search the space of possible terms until the first solution is found, thereby biasing their results towards the shortest or simplest solution as opposed to the most efficient. Synquid particularly suffers from this in cases where a more efficient solution would require at least as many syntax tree nodes as a less efficient implementation. As a result, Synquid and other synthesis methods are limited in their ability to generate programs for practical usage.

In this thesis, I alleviate this efficiency problem by combining type-directed program synthesis with type-based resource analysis. In particular, I use the method of resource analysis that is implemented in the programming language Resource Aware ML by Hoffmann, et. al [11]. This technique provides quantitative resource bounds as part of the type checking process by associating terms being type-checked with their cost

of evaluation. Moreover, the analysis is parametric in the resource of interest. This enables many different metrics to be analyzed, including heap space, clock cycles, evaluation steps, and more. Since this system is type-based, it cleanly complements and improves Synquid.

As an example, we consider the following efficient implementation of `compress` in RAML, which uses OCaml syntax:

```
let rec compress xs =
  match xs with
    | [] -> []
    | x3::x4 ->
    match compress x4 with
      [] -> [x3]
      x10::x11 ->
        if x3 == x10
          then x10::x11
          else x3::x10::x11
```

When analyzed with the number of `cons` operations as the analysis metric, the RAML compiler produces the type $L^2(\alpha) \xrightarrow{0/0} L^0(\alpha)$; this means that `compress` is a function that takes in a list with elements of any comparable type α as its argument, returns a list of the same type and, in order to run, takes 2 steps for each element in the input list (as an upper bound). For this function, as well as any RAML function for which a bound can be produced, these resource annotations for the type can be automatically and efficiently derived using LP solving. RAML is also able to derive univariate and multivariate polynomial bounds. However, it is not able to derive a bound for the earlier implementation since it is uses exponentially many `cons` operations.

This thesis presents several core contributions, which will be expanded on throughout:

- We have formalized the first resource-aware big-step operational semantics for the Synquid language.
- We have established the first bottom-up refinement type system that is amenable to automatic amortized resource analysis.
- We have shown the type system sound with respect to the operational semantics, thereby showing that the language is type-safe and that any bound derived for a given program is correct.
- We have developed resource-aware bidirectional and round-trip type systems that facilitate program synthesis, and have shown them sound and complete with respect to the bottom-up system.
- We have demonstrated that, given a resource-annotated type specification, if our system synthesizes a program, it will fulfill both the functional and cost requirements specified.

Chapter 2

Background

In this section, we will briefly discuss the ideas that form the foundation of our contributions. We will first discuss type-directed program synthesis, and then cover automatic amortized resource analysis. For each, we aim to give a brief introduction to previous work and describe our contributions.

2.1 Type-Directed Program Synthesis

As discussed in Chapter 1, many approaches to denoting specifications for program synthesis currently exist. Many take into account type information and enrich it with other details, but some exclusively rely on the type itself. The usual notion of a program type is frequently too coarse for such a role, so more descriptive types must be used. In particular, *refinement types* [8, 21], which augment base types with predicates from a decidable logic that restrict the possible inhabitants of that type, work well. As an example, let us look at the refinement type specification for the function `replicate`.

$$\texttt{replicate} : \quad n{:}\texttt{Nat} \rightarrow x{:}\alpha \rightarrow \{\texttt{List } \alpha \texttt{ | len } \nu = n\}$$

Here, our function takes in a natural number n and a value of some type α and returns a list that contains elements of type α. Moreover, the refinement of the codomain of this function specifies that the length of the resulting list must exactly match the number given. Since the only available value of type α is `x`, this type precisely specifies the desired function. We note that α may itself be instantiated with a refinement type, so at that instantiation, every element of the output list may be restricted to share some property with `x` that is specified by that particular type.

One particular formulation of refinement types, called *liquid types* [21, 23], specifies the decidable logic in such a way that reasoning about properties of programs based on their types can be automated for many

possible programs. Given a program, the liquid type inference algorithm can automatically infer a precise refinement type that verifiably describes the program.

However, when synthesizing a program based on its type, the usual inference algorithm is insufficient. In particular, when attempting to synthesize a program from a specification, the algorithm should be able to propagate information from the top level down to the individual pieces of the program, checking that each generated subexpression satisfies the type. This is the opposite of the liquid type inference algorithm, which constructs the type of a given expression from the known types of its subexpressions.

Polikarpova, et. al [20] propose a new approach, called *local liquid type checking*, which combines top-down and bottom-up type inference approaches. This technique is based on bidirectional type checking [19], which starts from a top-level type specification and interleaves bottom-up and top-down information in order to locally check parts of the program. However, their new procedure extends this to also propagate top-down information to inform the bottom-up inference, thereby enabling local checking of function applications, which are particularly difficult in the context of synthesis. This style of type-checking is titled *round-trip type checking*, which, in addition to other developments, allows local liquid type checking to function properly. Given the round-trip type checking system, Polikarpova, et. al [20] also define a synthesis algorithm based on the work of Osera and Zdancewic [18] that generates a correct program.

In previous work [21, 20], refinement type systems have been proven sound in regards to a small-step operational semantics. In order to track resource usage in a way amenable to automatic amortized resource analysis, we developed a big-step version of the operational semantics and showed that a round-trip type system with minimal differences from Synquid is sound in regards to the new semantics. Furthermore, we extended the round-trip type system to not only specify functional properties of well-typed programs but also resource usage constraints, thereby enabling it to synthesize programs that fit those restrictions.

2.2 Automatic Amortized Resource Analysis

In purely functional programs, reasoning about heap-based data structures is feasible since there are strong type guarantees on the shape of data. For example, we know that a functional list is immutable and does not have cycles. As a result, there are several techniques that can automatically or semi-automatically derive bounds that depend on the sizes of functional data structures. Automation is often achieved by relying on type systems [22, 4, 17, 3], recurrence relations [1, 9, 6, 5], and automatic amortized resource analysis [13, 14, 11, 2, 12]. For this work, we leverage the type-directed nature of automatic amortized resource analysis to specify resource usage restrictions on synthesized programs.

The core idea of automatic amortized resource analysis, or AARA, is to annotate each program point with

a *potential function* which maps sizes of reachable data structures to non-negative numbers. The potential functions must ensure that, for every input and every possible evaluation, the potential at a program point is sufficient to pay for the resource cost of the following transition and the potential at the next point. It then follows that the initial potential function describes an upper bound on the resource consumption of the program.

In order to determine the resource usage of a particular expression, the type derivation of that expression is augmented with resource usage annotations, which in turn establish a system of equations. This is then given to an LP solver, which efficiently minimizes the system and in turn produces an upper bound on the resource usage of the program. These equations are defined with respect to constants defined by a *resource metric* that allows the analysis system to specify the particular resource of interest. Because the system is parametric in terms of the resource, AARA can handle a range of quantities, including heap space, evaluation steps, and clock cycles, as well as any custom metric the user may wish to define. As an example, consider the following function in OCaml syntax:

```
let f (l1,l2,l3) =
          let t = append (l1,l2) in append (t,l3) end
```

Suppose we want to analyze the heap usage of this program. Furthermore, assume that when the first argument to `append` is a list of length n, the `append` function performs n `cons` operations, each of which consumes two heap cells, thereby consuming $2n$ heap cells in all. We would specify this with a set of type annotations by saying that `append` has the type $(L^{2+c}(T), L^{0+c}(T)) \xrightarrow{0+c'/0+c'} L^{0+c}(T)$, where c and c' are arbitrary constant offsets.

Using this setup, we can now analyze the function `f`. Firstly, we see that `append` is applied to `t` and `l3`. Therefore, for this to be well-typed, `t` must have the type $L^2(T)$ and `l3` must have the type $L^0(T)$. Next, we see that `t` is the result of `append (l1,l2)`. Therefore, for `t` to have the correct type, `l1` must have the type $L^4(T)$ and `l2` must have the type $L^2(T)$. From these annotations, our type system determines that this program consumes $4n + 2m$ heap cells, where n is the length of `l1` and m is the length of `l2`, and assigns `f` the type $(L^4(T), L^2(T), L^0(T)) \xrightarrow{0/0} L^0(T)$.

For this thesis, we formalize the first integration of AARA in a refinement type system. Furthermore, while previous versions of AARA require the program to be in *let-normal form*, which requires subexpressions to be bound to variables as frequently as possible, our system allows for programs to have a more general structure. In addition, other versions require well-typed programs to only use each variable at most once in the program in order to properly track resource usage. In this work, we enable variables to be used arbitrarily often due to a new mechanism for introducing equations for the LP solver.

Chapter 3

Technical Development

In this chapter, we develop the language, including its syntax, operational semantics, and type system. We then show that the type system is sound with respect to the operational semantics and that the annotations correctly calculate the resource usage of a well-typed program. Lastly, we prove that the three type systems presented here are sound and complete with respect to each other.

3.1 Language Definition

3.1.1 Syntax

The syntax of our language is presented in Figure 1. We distinguish between terms of the refinement language ψ and of the program language t. Terms in the former have sorts Δ; those with the Boolean sort $\mathbb{B}$ are called formulae. Terms of the latter are split into introduction and elimination terms (or I-terms and E-terms). The former propagate type information top-down by decomposing a complex requirement into simpler requirements for the components, and the latter propagate type information bottom-up by composing a complex property from the properties of the components. I-terms are further divided into branching terms and function terms; we disallow branching terms as arguments to function application to enable precise and efficient local type-checking. Our language focuses on basic language constructs, including constants, branching terms, and recursive and non-recursive functions.

Types are either scalar types (base types refined with a formula), dependent function types, or existential types. Our base types are Booleans, Integers, and the list type $L(T)$, where T is some other type. Our existential types follow their usage by Knowles and Flanagan [16] and replace the contextual types used by Polikarpova, et. al [20]. These allow the codomain of a function type to precisely mention the type of the

$\psi ::=$		*Refinement term:*
	$\mid \top \mid \bot \mid 0 \mid + \mid \ldots$ *(varies)*	interpreted symbol
	$\mid x$	uninterpreted symbol
	$\mid \psi\ \psi$	application
$\Delta ::=$		*Sort:*
	$\mid \mathbb{B} \mid \mathbb{Z} \mid \ldots$ *(varies)*	interpreted
	$\mid \delta$	uninterpreted
$t ::=$	$e \mid b \mid f$	*Program term*
$e ::=$		*E-term*
	$\mid x$	variable
	$\mid e\ e$	application
	$\mid n$	number
	$\mid$ `true` $\mid$ `false`	boolean constants
	$\mid$ `nil`	list-empty
	$\mid$ `cons`(e, e)	list-cons
$b ::=$		*Branching term*
	$\mid$ if e then t else t	conditional
	$\mid$ `match`$(e; t; x_1, x_2.t)$	match
$f ::=$		*Function term*
	$\mid \lambda x.t$	abstraction
	$\mid$ fix $x_1.\lambda x_2.t$	fixpoint
$B ::=$		*Base type:*
	$\mid$ Bool $\mid$ Int	primitive
	$\mid L(T)$	list
$T ::=$		*Type:*
	$\mid \{B \mid \psi\}$	scalar type
	$\mid x : T \to T$	dependent function type
	$\mid \exists x{:}T.T$	existential type

Figure 1: Syntax

$$\frac{\Delta \vdash_{\mathbb{Q}} v : \{B \mid \psi\}}{\Delta, x{:}\{B \mid \psi\} \vdash_{\mathbb{Q}} v : \{B \mid \nu = x\}}\ \text{(V:Self)}$$

$$\frac{n \in \mathbb{Z} \qquad \psi' = [n/\nu]\psi \qquad \mathsf{Valid}([\![\Delta]\!]_{\psi'} \Rightarrow \psi')}{\Delta \vdash_{\mathbb{Q}} n : \{\mathsf{Int} \mid \psi\}}\ \text{(V:Nat)} \quad \frac{\psi' = [\mathtt{nil}/\nu]\psi \qquad \mathsf{Valid}([\![\Delta]\!]_{\psi'} \Rightarrow \psi')}{\Delta \vdash_{\mathbb{Q}} \mathtt{nil} : \{L^q(T') \mid \psi\}}\ \text{(V:Nil)}$$

$$\frac{\psi' = [\top/\nu]\psi \qquad \mathsf{Valid}([\![\Delta]\!]_{\psi'} \Rightarrow \psi')}{\Delta \vdash_{\mathbb{Q}} \mathtt{true} : \{\mathsf{Bool} \mid \psi\}}\ \text{(V:True)} \quad \frac{\psi' = [\bot/\nu]\psi \qquad \mathsf{Valid}([\![\Delta]\!]_{\psi'} \Rightarrow \psi')}{\Delta \vdash_{\mathbb{Q}} \mathtt{false} : \{\mathsf{Bool} \mid \psi\}}\ \text{(V:False)}$$

$$\frac{\Delta \vdash_{\mathbb{Q}} V : \Gamma \qquad \Gamma, x : T_x; \Delta, x : |T_x| \vdash^{\frac{q}{q'}}_{\mathbb{Q}} t : T \qquad \Gamma \Downarrow}{\Delta \vdash_{\mathbb{Q}} (\lambda x.t, V) : T_x \xrightarrow{q/q'} T}\ \text{(V:Lam)}$$

$$\frac{\begin{array}{c}\Delta \vdash_{\mathbb{Q}} v_1 : T' \qquad \Delta, x_1{:}|T'| \vdash_{\mathbb{Q}} v_2 : \{L^q(T') \mid \psi_{\mathtt{cons}}\} \\ \psi'' = [\overline{\mathtt{cons}(v_1, v_2)}/\nu]\psi \qquad \mathsf{Valid}([\![\Delta]\!]_{\psi''} \Rightarrow \psi'')\end{array}}{\Delta \vdash_{\mathbb{Q}} \mathtt{cons}(v_1, v_2) : \{L^q(T') \mid \psi\}}\ \text{(V:Cons)}$$

$$\frac{\Delta \vdash_{\mathbb{Q}} v' : T_1 \qquad \Delta \vdash_{\mathbb{Q}} v : [\overline{v'}/x]T_2}{\Delta \vdash_{\mathbb{Q}} v : \exists x{:}T_1.T_2}\ \text{(V:Wit)} \quad \frac{\Delta \vdash_{\mathbb{Q}} v : T}{\Delta, \psi \vdash_{\mathbb{Q}} v : T}\ \text{(V:BWeak)} \quad \frac{\Delta \vdash_{\mathbb{Q}} v : T}{\Delta, x{:}T' \vdash_{\mathbb{Q}} v : T}\ \text{(V:Weak)}$$

Figure 2: Value Checking Rules

$$\frac{}{\Delta \vdash_{\mathbb{Q}} V : \emptyset}\ \text{(VC:Em)} \quad \frac{\Delta \vdash_{\mathbb{Q}} V : \Gamma \qquad \Delta \vdash_{\mathbb{Q}} v : T}{\Delta, x{:}T \vdash_{\mathbb{Q}} V[x \mapsto v] : \Gamma, x{:}T}\ \text{(VC:Ext)}$$

$$\frac{\Delta \vdash_{\mathbb{Q}} V : \Gamma \qquad \Delta \vdash_{\mathbb{Q}} T}{\Delta, x{:}|T| \vdash_{\mathbb{Q}} V : \Gamma}\ \text{(VC:DExt)} \quad \frac{\Delta \vdash_{\mathbb{Q}} V : \Gamma}{\Delta, \psi \vdash_{\mathbb{Q}} V : \Gamma}\ \text{(VC:BExt)}$$

Figure 3: Well-Formed Context Rules

argument after application without substituting program terms into our refinement language.

3.1.2 Big-Step Operational Semantics

In the semantics of our language, we distinguish between expressions, which may need to be evaluated, and values, which have been evaluated. Values are described as follows.

$$v ::= n \mid \mathtt{true} \mid \mathtt{false} \mid (\lambda x.t, V) \mid \mathtt{nil} \mid \mathtt{cons}(v_1, v_2)$$

Figure 2 defines the judgment $\Delta \vdash_{\mathbb{Q}} v : T$, which states that the value v has type T, where all free variables mentioned in T are bound in Δ. We thus say that an environment and context are well-formed with respect to each other with $\Delta \vdash_{\mathbb{Q}} V : \Gamma$ following the rules in Figure 3. We note that the rules in Figure 2 primarily check that the value v satisfies the refinement in T. To support this, we define a way to substitute program values into the refinement language in Figure 4.

$$
\begin{array}{rcl}
[\overline{n}/x]\psi & = & [n/x]\psi \\
[\overline{\texttt{true}}/x]\psi & = & [\top/x]\psi \\
[\overline{\texttt{false}}/x]\psi & = & [\bot/x]\psi \\
[\overline{\texttt{nil}}/x]\psi & = & [\texttt{nil}/x]\psi \\
[\overline{\texttt{cons}(v_1, v_2)}/x]\psi & = & [\overline{v_1}, \overline{v_2}/x_1, x_2]([\texttt{cons}(x_1, x_2)/x]\psi) \\
[\overline{(\lambda y.t, V)}/x]\psi & = & \psi
\end{array}
$$

Figure 4: Value Translation

$$\frac{x \in \mathrm{dom}(V)}{V \vdash x \Downarrow V(x) \mid K^{\mathrm{var}}}\ (\text{E:Var}) \qquad \frac{}{V \vdash (\lambda x.t, V) \Downarrow (\lambda x.t, V) \mid K^{\mathrm{fun}}}\ (\text{E:Fun})$$

$$\frac{n \in \mathbb{N}}{V \vdash n \Downarrow n \mid K^{\mathrm{int}}}\ (\text{E:Nat}) \qquad \frac{b \in \{\texttt{true}, \texttt{false}\}}{V \vdash b \Downarrow b \mid K^{\mathrm{bool}}}\ (\text{E:Bool}) \qquad \frac{}{V \vdash \texttt{nil} \Downarrow \texttt{nil} \mid K^{\mathrm{nil}}}\ (\text{E:Nil})$$

$$\frac{V \vdash e_1 \Downarrow v_1 \mid (q, q') \qquad V \vdash e_2 \Downarrow v_2 \mid (p, p')}{V \vdash \texttt{cons}(e_1, e_2) \Downarrow \texttt{cons}(v_1, v_2) \mid K_1^{\mathrm{cons}} \cdot (q, q') \cdot K_2^{\mathrm{cons}} \cdot (p, p') \cdot K_3^{\mathrm{cons}}}\ (\text{E:Cons})$$

$$\frac{V \vdash e_1 \Downarrow (\lambda x.t', V') \mid (q, q') \qquad V \vdash t \Downarrow v \mid (p, p') \qquad V'[x \mapsto v] \vdash t' \Downarrow v' \mid (r, r')}{V \vdash e\ t \Downarrow v' \mid K_1^{\mathrm{app}} \cdot (q, q') \cdot K_2^{\mathrm{app}} \cdot (p, p') \cdot K_3^{\mathrm{app}} \cdot (r, r') \cdot K_4^{\mathrm{app}}}\ (\text{E:App})$$

$$\frac{V \vdash e \Downarrow \texttt{true} \mid (q, q') \qquad V \vdash t_1 \Downarrow v \mid (p, p')}{V \vdash \mathsf{if}\ e\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2 \Downarrow v \mid K^{\mathrm{con}} \cdot (q, q') \cdot K_1^{\mathrm{conT}} \cdot (p, p') \cdot K_2^{\mathrm{conT}}}\ (\text{E:CondT})$$

$$\frac{V \vdash e \Downarrow \texttt{false} \mid (q, q') \qquad V \vdash t_2 \Downarrow v \mid (p, p')}{V \vdash \mathsf{if}\ e\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2 \Downarrow v \mid K^{\mathrm{con}} \cdot (q, q') \cdot K_1^{\mathrm{conF}} \cdot (p, p') \cdot K_2^{\mathrm{conF}}}\ (\text{E:CondF})$$

$$\frac{V \vdash e \Downarrow \texttt{nil} \mid (q, q') \qquad V \vdash t_1 \Downarrow v \mid (p, p')}{V \vdash \texttt{match}(e; t_1; x_1, x_2.t_2) \Downarrow v \mid K^{\mathrm{mat}} \cdot (q, q') \cdot K_1^{\mathrm{matN}} \cdot (p, p') \cdot K_2^{\mathrm{matN}}}\ (\text{E:MatchN})$$

$$\frac{V \vdash e \Downarrow \texttt{cons}(v_1, v_2) \mid (q, q') \qquad V[x_1 \mapsto v_1, x_2 \mapsto v_2] \vdash t_2 \Downarrow v \mid (p, p')}{V \vdash \texttt{match}(e; t_1; x_1, x_2.t_2) \Downarrow v \mid K^{\mathrm{mat}}(q, q') \cdot K_1^{\mathrm{matC}} \cdot (p, p') \cdot K_2^{\mathrm{matC}}}\ (\text{E:MatchC})$$

$$\frac{t' = [\mathsf{fix}\ f.\lambda x.t/f]e}{V \vdash \mathsf{fix}\ f.\lambda x.t \Downarrow (\lambda x.t', V) \mid K^{\mathrm{fix}}}\ (\text{E:Fix})$$

Figure 5: Operational Semantics

$$\frac{\mathsf{Shape}(T) \in \{\alpha, \mathsf{Bool}, \mathsf{Int}\}}{T \curlyvee (T, T)}\ \curlyvee\text{-SELF} \qquad \frac{p = q + r}{\{L^p(T) \mid \psi\} \curlyvee (\{L^q(T) \mid \psi\}, \{L^r(T) \mid \psi\})}\ \curlyvee\text{-SPLIT}$$

$$\frac{T \curlyvee (T_1, T_2)}{\exists x{:}T'.T \curlyvee (\exists x{:}T'.T_1, \exists x{:}T'.T_2)}\ \curlyvee\text{-LET}$$

$$\frac{}{\emptyset \Downarrow}\ \Gamma\text{-EMPTY} \qquad \frac{\Gamma \Downarrow \qquad T \curlyvee (T, T)}{\Gamma, x : T \Downarrow}\ \Gamma\text{-EXTEND}$$

$$\begin{aligned}
\Phi(\mathtt{cons}(v_1, v_2) : \{L^q(T) \mid \psi\}) &= q + \Phi(v_1 : T) + \Phi(v_2 : \{L^q(T) \mid \psi_{\mathtt{cons}}\}) \\
\Phi(v : \exists x{:}T_1.T_2) &= \Phi(v : T_2) \\
\Phi(v : T) &= 0 \text{ otherwise} \\
\Phi(V : \Gamma) &= \sum_{x \in \mathrm{dom}(\Gamma)} \Phi(V(x) : \Gamma(x))
\end{aligned}$$

Figure 6: Rules Governing Potential

We now define the following evaluation judgement by the rules in Figure 5:

$$V \vdash t \Downarrow v \mid (q, q')$$

This means that under the environment V that maps variables to values, the term t evaluates to the value v with $q \in \mathbb{Q}_0^+$ resource units available before evaluation and $q' \in \mathbb{Q}_0^+$ available after. We can identify a single $q \in \mathbb{Q}$ with these costs by considering $q \geq 0$ to stand for $(q, 0)$ and $q < 0$ for $(0, -q)$. This allows us to frequently represent constant costs by $K_i^{\mathrm{k}} : K \times \mathbb{N} \to \mathbb{Q}$, where $k \in K$ is a member of a set of constants that describe terms. This mapping allows us to avoid case distinctions on the sign of these costs. We use the following proposition in cases where manipulation of watermarks of the form (q, q') is required.

Proposition 1. *Let* $(q, q') = (r, r') \cdot (s, s')$.

1. $q \geq r$ *and* $q - q' = r - r' + s - s'$
2. *If* $(p, p') = (\overline{r}, r') \cdot (s, s')$ *and* $\overline{r} \geq r$ *then* $p \geq q$ *and* $p' = q'$
3. *If* $(p, p') = (r, r') \cdot (\overline{s}, s')$ *and* $\overline{s} \geq s$ *then* $p \geq q$ *and* $p' \leq q'$
4. $(r, r') \cdot ((s, s') \cdot (t, t')) = ((r, r') \cdot (s, s')) \cdot (t, t')$

3.1.3 Type Systems

We define the following typing judgement with the rules in Figure 9 on page 18:

$$\Gamma; \Delta \vdash^{q}_{q'}{}_{\mathbb{Q}} t : T$$

$$\frac{}{\emptyset \mid\mid \emptyset; \emptyset} \text{ (}\Gamma\text{:Emp)} \qquad \frac{\Gamma \mid\mid \Gamma_1; \Gamma_2 \qquad T \curlyvee (T_1, T_2)}{\Gamma, x{:}T \mid\mid \Gamma_1, x{:}T_1; \Gamma_2, x{:}T_2} \text{ (}\Gamma\text{:Ext)}$$

Figure 7: Context Splitting

$$\begin{array}{rcl} |\mathsf{Int}| & = & \mathsf{Int} \\ |\mathsf{Bool}| & = & \mathsf{Bool} \\ |L^q(T)| & = & L(|T|) \\ |\{B \mid \psi\}| & = & \{|B| \mid \psi\} \\ |x{:}T_x \xrightarrow{q/q'} T| & = & x{:}|T_x| \to |T| \\ |\exists x{:}T_1.T_2| & = & \exists x.T_1.|T_2| \end{array}$$

Figure 8: Potential Stripping

Here, the typing environment Γ maps variables to resource-annotated types and the guard environment Δ maps both variables to types (that are not resource-annotated) and contains boolean formulae called *path conditions*. Additionally, t is a term, and T is a resource-annotated type. The judgement means that if there are at least $q + \Phi(\Gamma)$ resource units available, the term t is of type T and can be evaluated. If it evaluates to a value v, then there are $q' + \Phi(v{:}T)$ resource units left over. Φ and other constructs involved in describing potential are defined in Figure 6.

Since AARA relies on affine usage of program variables for proper resource tracking, we use two ordered contexts: Γ, which is affine, and Δ, which is unrestricted. Since refinements do not affect the resource consumption of a program, we distinguish between *uses* of a variable, which occur as program terms and are bound in Γ, and *mentions* of a variable, which appear in the refinements and are bound in Δ. As a result, we can freely mention a given variable as often as needed in the refinement language, but must be more careful with program variables. This differs from previous work [20], which treats uses and mentions as the same. To properly handle the affine context, we use the context splitting judgement, defined by the rules in Figure 7, to copy Γ in a way that soundly splits the potential carried by each variable in rules where the typing environment must be split. This allows programs to reuse variables even though the context itself is affine, thereby maintaining the same expressivity as previous systems. Additionally, we use the operation $|\cdot|$ defined in Figure 8 to change resource annotated types into unannotated types, and we use the judgement $\Delta \vdash_{\mathbb{Q}} T$ as defined in Figure 10 to mean that every variable in T is bound in Δ with the correct sort. Lastly, we use the rules in Figure 11 to define the judgement $\vdash_{\mathbb{Q}} \Delta$, which means that every variable bound in Δ is well-formed in the context of all preceding bindings.

Our type rules refer to the subtyping rules presented in Figure 12. Most of these rules serve to decompose the type; in the case of the rule Sub:Sc, though, subtyping is reduced to implication between the refinements.

Specifically, the guard environment is converted into a conjunction of assumptions in the following manner.

$$[\![\Delta]\!]_\psi = \mathsf{P}(\Delta) \wedge \mathsf{B}_{\mathsf{FV}(\mathsf{P}(\Delta)) \cup \mathsf{FV}(\psi)}(\Delta)$$

where

$$\mathsf{B}_v(\Delta; x{:}\{B \mid \psi\}) = \begin{cases} [x/\nu]\psi \wedge \mathsf{B}_{v \setminus \{x\} \cup \mathsf{FV}(\psi)}(\Delta) & (x \in v) \\ \mathsf{B}_v(\Delta) & (\text{otherwise}) \end{cases}$$

$$\mathsf{B}_v(\Delta; x{:}T) = \mathsf{B}_v(\Delta) \qquad (T \text{ non-scalar})$$

$$\mathsf{B}_v(\cdot) = \top$$

Informally, $[\![\Delta]\!]_\psi$ provides a conjunction of all of the path conditions in Δ, as well as all of the formulae associated with the transitive closure of the free variables of ψ. We note that our notion of subtyping preserves the usual desired properties of subtyping, including reflexivity and transitivity. In addition, it has been shown by Flanagan [8] that the values of a given type are inhabitants of all possible supertypes.

In order to specify properties of lists in our refinement language, we parameterize our system with boolean formulae $g_{\texttt{nil}}$, $g_{\texttt{cons}}$, and $\psi_{\texttt{cons}}$ that we call *guard formulae*. The first describes a standard property of the empty list, the second designates a standard property of a non-empty list, and the third relates the head of a list to the tail. To do so, $g_{\texttt{cons}}$ can mention x_1 and x_2, which represent the head and tail of a non-empty list, and $\psi_{\texttt{cons}}$ can mention x_1, which stands for the head of the list. We note that $g_{\texttt{nil}}$ and $g_{\texttt{cons}}$ must be instantiated with sufficiently general formulae that can be satisfied by any possible list. For example, one common usage is as follows, where $g_{\texttt{nil}}$ and $g_{\texttt{cons}}$ provide information about the length of the list they refine, and given a refinement level function heads, which maps a list to the set containing its head, $\psi_{\texttt{cons}}$ requires that the head of a `cons` operation be less than the next element in the list, thereby specifying a sorted list:

$$g_{\texttt{nil}} = (\mathsf{len}\ \nu = 0)$$

$$g_{\texttt{cons}} = (\mathsf{len}\ \nu = 1 + \mathsf{len}\ x_2)$$

$$\psi_{\texttt{cons}} = (x_1 < \mathsf{heads}\ \nu)$$

We also note that our language includes dependent function types $x{:}T_x \to T$, where x may appear free in T. However, the LT:App rule only considers non-dependent function types $T_x \to T$. As discussed in Knowles, et. al [16], the following rule for dependent function types is admissible, given the fact that subtyping does not affect resource consumption:

$$\frac{\Gamma \mid\mid \Gamma_1; \Gamma_2 \qquad \Gamma_1; \Delta \left|\frac{q - K_1^{\text{app}}}{p}\right._{\mathbb{Q}} e : x{:}T_x \xrightarrow{r - K_3^{\text{app}}/q' + K_4^{\text{app}}} T \qquad \Gamma_2; \Delta \left|\frac{p - K_2^{\text{app}}}{r}\right._{\mathbb{Q}} t : T'_x \qquad \Delta \vdash T'_x <: T_x}{\Gamma \left|\frac{q}{q'}\right._{\mathbb{Q}} e\ t : (\exists x{:}|T'_x|.T)} \text{ (LT:AppDeriv)}$$

We prove two main claims about our bottom-up type system. First, we argue in Section 3.2.1 that if a well-typed term t evaluates to a value v in a well-formed environment, then that value checks against type T. This claim ensures that every well-typed term evaluates to a value that satisfies the refinement of its type. Second, we show in Section 3.2.2 that the type derivation computes a correct bound. In particular, we claim that if a well-typed term t evaluates to a value v in a well-formed environment, then the initial potential of the context gives an upper bound on the watermark of the actual resource usage. Furthermore, we claim that the difference between the initial and final potential gives an upper bound on the actual resource consumption.

In addition to our bottom-up type rules, we define a bidirectional and a round-trip version of the rules, respectively shown in Figures 13 and 14. The bidirectional rules use two judgements: an inference judgement $\Gamma; \Delta \left|_{q'}^{q}\right._{\mathbb{Q}} e \uparrow_b T$, which states that the term e generates the type T; and a checking judgement $\Gamma; \Delta \left|_{q'}^{q}\right._{\mathbb{Q}} t \downarrow_b T$, stating that the term t checks against a known type T. Similarly, the round-trip system uses two judgements: the old checking judgement $\Gamma; \Delta \left|_{q'}^{q}\right._{\mathbb{Q}} t \downarrow T$; and a strengthening judgement $\Gamma; \Delta \left|_{q'}^{q}\right._{\mathbb{Q}} e \downarrow T \uparrow T'$, which means that e checks against a known type T and generates a stronger type T' (that is, $\Delta \vdash T' <: T$). In Section 3.2.3, we show that all three systems are connected by proving that the bidirectional and bottom-up systems are sound and complete with respect to each other and then showing that the bidirectional and round-trip systems are sound and complete with respect to each other.

3.2 Proofs

Before proving the key theorems of our system, we establish a few lemmas for later use.

Lemma 1. *Let T, T' be two resource-annotated refinement types with $T <: T'$. Then $\Phi(v{:}T) \geq \Phi(v{:}T')$ for all $v \in$ Val such that $\Delta \vdash_{\mathbb{Q}} v{:}T$ and $\Delta \vdash_{\mathbb{Q}} v{:}T'$.*

Proof. We go by induction on the definition of Φ. Since refinements do not affect potential or resource usage, it follows by the same argument as in [10]. □

Lemma 2. *If $\Delta \vdash_{\mathbb{Q}} V : \Gamma$ and $\Gamma \mid\mid \Gamma_1; \Gamma_2$ then $\Delta \vdash_{\mathbb{Q}} V : \Gamma_1$ and $\Delta \vdash_{\mathbb{Q}} V : \Gamma_2$.*

Proof. We go by induction on the rules defining $\Gamma \mid\mid \Gamma_1; \Gamma_2$. In the base case, all three are empty, so the claim holds trivially. Otherwise, we have by assumption that $\Delta \vdash_{\mathbb{Q}} V(x) : T$ where $\Gamma(x) = T$. Since the value checking rules do not take potential annotations into account, we have that $\Delta \vdash_{\mathbb{Q}} V(x) : T_1$ and $\Delta \vdash_{\mathbb{Q}} V(x) : T_2$ where $T \curlyvee (T_1, T_2)$. Thus, the claim holds. □

$$\frac{}{\Gamma, x{:}\{B \mid \psi\}; \Delta, x{:}\{|B| \mid \psi\} \left|\frac{q+K^{\text{var}}}{q}\right._{\mathbb{Q}} x : \{B \mid \nu = x\}} \text{ (LT:Var-Base)}$$

$$\frac{T \text{ is not a base type}}{\Gamma, x{:}T; \Delta, x{:}|T| \left|\frac{q+K^{\text{var}}}{q}\right._{\mathbb{Q}} x : T} \text{ (LT:Var)}$$

$$\frac{}{\emptyset; \Delta \left|\frac{q+K^{\text{bool}}}{q}\right._{\mathbb{Q}} \texttt{true} : \{\mathsf{Bool} \mid \nu = \top\}} \text{ (LT:True)} \quad \frac{}{\emptyset; \Delta \left|\frac{q+K^{\text{bool}}}{q}\right._{\mathbb{Q}} \texttt{false} : \{\mathsf{Bool} \mid \nu = \bot\}} \text{ (LT:False)}$$

$$\frac{}{\emptyset; \Delta \left|\frac{q+K^{\text{int}}}{q}\right._{\mathbb{Q}} n : \{\mathsf{Int} \mid \nu = n\}} \text{ (LT:Int)} \quad \frac{}{\emptyset; \Delta \left|\frac{q+K^{\text{nil}}}{q}\right._{\mathbb{Q}} \texttt{nil} : \{L^p(T) \mid g_{[]}\}} \text{ (LT:Nil)}$$

$$\frac{\begin{array}{c}\Gamma \mid\mid \Gamma_1; \Gamma_2 \qquad \Gamma_1; \Delta \left|\frac{q-K_1^{\text{cons}}}{p}\right._{\mathbb{Q}} e_1 : T \\ \Gamma_2; \Delta, x_1{:}|T| \left|\frac{p-K_2^{\text{cons}}}{q'+r+K_3^{\text{cons}}}\right._{\mathbb{Q}} e_2 : \{L^r(T) \mid \psi_{\texttt{cons}}\}\end{array}}{\Gamma; \Delta \left|\frac{q}{q'}\right._{\mathbb{Q}} \texttt{cons}(e_1, e_2) : \exists x_1{:}|T|, x_2{:}\{L(|T|) \mid \psi_{\texttt{cons}}\}.\{L^r(T) \mid g_{\texttt{cons}}\}} \text{ (LT:Cons)}$$

$$\frac{\Delta \vdash_{\mathbb{Q}} (x{:}T_x \to T) \qquad \Gamma, x{:}T_x; \Delta, x{:}|T_x| \left|\frac{p}{p'}\right._{\mathbb{Q}} t : T \qquad \Gamma \Downarrow}{\Gamma; \Delta \left|\frac{q+K^{\text{fun}}}{q}\right._{\mathbb{Q}} \lambda x.t : (x : T_x \xrightarrow{p/p'} T)} \text{ (LT:Fun)}$$

$$\frac{\Gamma \mid\mid \Gamma_1; \Gamma_2 \qquad \Gamma_1; \Delta \left|\frac{q-K_1^{\text{app}}}{p}\right._{\mathbb{Q}} e : T_x \xrightarrow{r-K_3^{\text{app}}/q'+K_4^{\text{app}}} T \qquad \Gamma_2; \Delta \left|\frac{p-K_2^{\text{app}}}{r}\right._{\mathbb{Q}} t : T_x}{\Gamma; \Delta \left|\frac{q}{q'}\right._{\mathbb{Q}} e\ t : T} \text{ (LT:App)}$$

$$\frac{\begin{array}{c}\Delta \vdash_{\mathbb{Q}} T \qquad \Gamma \mid\mid \Gamma_1; \Gamma_2 \qquad \Gamma_1; \Delta \left|\frac{q-K^{\text{con}}}{p}\right._{\mathbb{Q}} e : \exists \bar{z}{:}\bar{T}_z.\{\mathsf{Bool} \mid \psi\} \\ \Gamma_2; \Delta, \bar{z}{:}\bar{T}_z, [\top/\nu]\psi \left|\frac{p-K_1^{\text{conT}}}{q'+K_2^{\text{conT}}}\right._{\mathbb{Q}} t_1 : T \qquad \Gamma_2; \Delta, \bar{z}{:}\bar{T}_z, [\bot/\nu]\psi \left|\frac{p-K_1^{\text{conF}}}{q'+K_2^{\text{conF}}}\right._{\mathbb{Q}} t_2 : T\end{array}}{\Gamma; \Delta \left|\frac{q}{q'}\right._{\mathbb{Q}} \mathsf{if}\ e\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2 : T} \text{ (LT:Cond)}$$

$$\frac{\begin{array}{c}\Delta \vdash_{\mathbb{Q}} T \qquad \Gamma \mid\mid \Gamma_1; \Gamma_2 \\ \Gamma_1; \Delta \left|\frac{q-K^{\text{mat}}}{r}\right._{\mathbb{Q}} e : \exists \bar{z}{:}\bar{T}_z.\{L^p(T') \mid \psi\} \qquad \Gamma_2; \Delta, \bar{z}{:}\bar{T}_z, [x'/\nu]g_{\texttt{nil}}, [x'/\nu]\psi \left|\frac{r-K_1^{\text{matN}}}{q'+K_2^{\text{matN}}}\right._{\mathbb{Q}} t_1 : T \\ \Delta' = \Delta, \bar{z}{:}\bar{T}_z, x_1 : |T'|, x_2 : \{L(|T'|) \mid \psi_{\texttt{cons}}\}, [x'/\nu]g_{\texttt{cons}}, [x'/\nu]\psi \\ \Gamma_2, x_1 : T', x_2 : \{L^p(T') \mid \psi_{\texttt{cons}}\}; \Delta' \left|\frac{r+p-K_1^{\text{matC}}}{q'+K_2^{\text{matC}}}\right._{\mathbb{Q}} t_2 : T\end{array}}{\Gamma; \Delta \left|\frac{q}{q'}\right._{\mathbb{Q}} \texttt{match}(e; t_1; x_1, x_2.t_2) : T} \text{ (LT:Match)}$$

$$\frac{\Delta \vdash_{\mathbb{Q}} T_x \xrightarrow{p/p'} T \qquad f{:}T_x \xrightarrow{p/p'} T, x{:}T; \Delta, x{:}|T| \left|\frac{p}{p'}\right._{\mathbb{Q}} t : T}{\emptyset; \Delta \left|\frac{q+K^{\text{fix}}}{q}\right._{\mathbb{Q}} \mathsf{fix}\ f.\lambda x.t : T_x \xrightarrow{p/p'} T} \text{ (LT:Fix)}$$

Figure 9: Rules for Liquid Type Checking (1 of 2)

$$\frac{\Gamma; \Delta \vdash^{q}_{q'\,\mathbb{Q}} t : T \qquad \Delta \vdash T <: T' \qquad \Delta \vdash_{\mathbb{Q}} T'}{\Gamma; \Delta \vdash^{q}_{q'\,\mathbb{Q}} t : T'} \text{ (LT:Sub)}$$

$$\frac{\Gamma; \Delta \vdash^{q}_{q'\,\mathbb{Q}} t : T}{\Gamma, x{:}T'; \Delta, x{:}|T'| \vdash^{q}_{q'\,\mathbb{Q}} t : T} \text{ (LT:Weaken)} \quad \frac{\Gamma; \Delta \vdash^{p}_{p'\,\mathbb{Q}} t : T \qquad q \geq p \qquad q - p \geq q' - p'}{\Gamma; \Delta \vdash^{q}_{q'\,\mathbb{Q}} t : T} \text{ (LT:Relax)}$$

Figure 9: Rules for Liquid Type Checking (2 of 2)

$$\frac{\Delta, \nu : B \vdash \psi}{\Delta \vdash \{B \mid \psi\}} \text{ (WF:Sc)} \quad \frac{\Delta \vdash T_1 \qquad \Delta, x{:}T_1 \vdash T_2}{\Delta \vdash \exists x{:}T_1.T_2} \text{ (WF:Ex)}$$

$$\frac{\Delta \vdash \{B \mid \psi\} \qquad \Delta, x{:}\{B \mid \psi\} \vdash T}{\Delta \vdash x{:}\{B \mid \psi\} \to T} \text{ (WF:FO)}$$

$$\frac{T_x \text{ non scalar} \qquad \Delta \vdash T_x \qquad \Delta \vdash T}{\Delta \vdash x{:}T_x \to T} \text{ (WF:HO)}$$

Figure 10: Well-Formed Types

$$\frac{}{\vdash_{\mathbb{Q}} \emptyset} \text{ (W}\Delta\text{:Emp)} \quad \frac{\vdash_{\mathbb{Q}} \Delta \qquad \Delta \vdash_{\mathbb{Q}} x : T}{\vdash_{\mathbb{Q}} \Delta, x{:}T} \text{ (W}\Delta\text{:Ext)} \quad \frac{\vdash_{\mathbb{Q}} \Delta}{\vdash_{\mathbb{Q}} \Delta, \psi} \text{ (W}\Delta\text{:Ext)}$$

Figure 11: Well-Formed Guard Environment

$$\frac{\Delta \vdash B <: B' \qquad \mathsf{Valid}\,(\llbracket \Delta \rrbracket_{\psi \Rightarrow \psi'} \wedge \psi \Rightarrow \psi')}{\Delta \vdash \{B \mid \psi\} <: \{B' \mid \psi'\}} \text{ (Sub:Sc)}$$

$$\frac{\Delta \vdash T_y <: T_x \qquad \Delta, y : T_y \vdash [y/x]T <: T'}{\Delta \vdash x : T_x \to T <: y : T_y \to T'} \text{ (Sub:Fun)}$$

$$\frac{\Delta \vdash T <: T' \qquad p \geq q}{\Delta \vdash L^p(T) <: L^q(T')} \text{ (Sub:List)} \quad \frac{B \in \{\alpha, \mathsf{Bool}, \mathsf{Int}\}}{\Delta \vdash B <: B} \text{ (Sub:Refl)}$$

$$\frac{\Delta \vdash v : T_1 \qquad \Delta \vdash T <: [v/x]T_2}{\Delta \vdash T <: \exists x{:}T_1.T_2} \text{ (Sub:Wit)} \quad \frac{\Delta, x{:}T_1 \vdash T_2 <: T \qquad \Delta \vdash T}{\Delta \vdash \exists x{:}T_1.T_2 <: T} \text{ (Sub:Bind)}$$

Figure 12: Subtyping Rules

$$\frac{}{\Gamma, x{:}\{B \mid \psi\}; \Delta \left|\frac{q+K^{\mathrm{var}}}{q}\right._{\mathbb{Q}} x \uparrow_b \{B \mid \nu = x\}} \text{(BD:Var-Base)} \qquad \frac{T \text{ is not a base type}}{\Gamma, x{:}T; \Delta \left|\frac{q+K^{\mathrm{var}}}{q}\right._{\mathbb{Q}} x \uparrow_b T} \text{(BD:Var)}$$

$$\frac{}{\emptyset; \Delta \left|\frac{q+K^{\mathrm{bool}}}{q}\right._{\mathbb{Q}} \mathtt{true} \uparrow_b \{\mathsf{Bool} \mid \nu = \top\}} \text{(BD:True)} \qquad \frac{}{\emptyset; \Delta \left|\frac{q+K^{\mathrm{bool}}}{q}\right._{\mathbb{Q}} \mathtt{false} \uparrow_b \{\mathsf{Bool} \mid \nu = \bot\}} \text{(BD:False)}$$

$$\frac{}{\emptyset; \Delta \left|\frac{q+K^{\mathrm{int}}}{q}\right._{\mathbb{Q}} n \uparrow_b \{\mathsf{Int} \mid \nu = n\}} \text{(BD:Int)} \qquad \frac{}{\emptyset; \Delta \left|\frac{q+K^{\mathrm{nil}}}{q}\right._{\mathbb{Q}} \mathtt{nil} \uparrow_b \{L^p(T) \mid g_{[]}\}} \text{(BD:Nil)}$$

$$\frac{\begin{array}{c}\Gamma \mid\mid \Gamma_1; \Gamma_2 \qquad \Gamma_1; \Delta \left|\frac{q-K_1^{\mathrm{cons}}}{p}\right._{\mathbb{Q}} e_1 \uparrow_b T \\ \Gamma_2; \Delta, x_1{:}|T| \left|\frac{p-K_2^{\mathrm{cons}}}{q'+r+K_3^{\mathrm{cons}}}\right._{\mathbb{Q}} e_2 \uparrow_b T' \qquad \Delta, x_1{:}|T| \vdash T' <: \{L^r(T) \mid \psi_{\mathtt{cons}}\}\end{array}}{\Gamma; \Delta \left|\frac{q}{q'}\right._{\mathbb{Q}} \mathtt{cons}(e_1, e_2) \uparrow_b \exists x_1{:}|T|, x_2{:}\{L(|T|) \mid \psi_{\mathtt{cons}}\}.\{L^r(T) \mid g_{\mathtt{cons}}\}} \text{(BD:Cons)}$$

$$\frac{\Gamma, y{:}T_x; \Delta, y{:}|T_x| \left|\frac{p}{p'}\right._{\mathbb{Q}} t \downarrow_b [y/x]T \qquad \Gamma \Downarrow}{\Gamma; \Delta \left|\frac{q+K^{\mathrm{fun}}}{q}\right._{\mathbb{Q}} \lambda y.t \downarrow_b (x : T_x \xrightarrow{p/p'} T)} \text{(BD:Fun)}$$

$$\frac{\begin{array}{c}\Gamma \mid\mid \Gamma_1; \Gamma_2 \qquad \Gamma_1; \Delta \left|\frac{q-K_1^{\mathrm{app}}}{p}\right._{\mathbb{Q}} e_1 \uparrow_b x{:}\{B \mid \psi\} \xrightarrow{r-K_3^{\mathrm{app}}/q'+K_4^{\mathrm{app}}} T \\ \Gamma_2; \Delta \left|\frac{p-K_2^{\mathrm{app}}}{r}\right._{\mathbb{Q}} e_2 \uparrow_b T_x \qquad \Delta \vdash T_x <: \{B \mid \psi\}\end{array}}{\Gamma; \Delta \left|\frac{q}{q'}\right._{\mathbb{Q}} e_1\ e_2 \uparrow_b \exists x{:}T_x.T} \text{(BD:AppFO)}$$

$$\frac{\Gamma \mid\mid \Gamma_1; \Gamma_2 \qquad \Gamma_1; \Delta \left|\frac{q-K_1^{\mathrm{app}}}{p}\right._{\mathbb{Q}} e \uparrow_b T_x \xrightarrow{r-K_3^{\mathrm{app}}/q'+K_4^{\mathrm{app}}} T \qquad \Gamma_2; \Delta \left|\frac{p-K_2^{\mathrm{app}}}{r}\right._{\mathbb{Q}} f \downarrow_b T_x}{\Gamma; \Delta \left|\frac{q}{q'}\right._{\mathbb{Q}} e\ f \uparrow_b T} \text{(BD:AppHO)}$$

$$\frac{\begin{array}{c}\Gamma \mid\mid \Gamma_1; \Gamma_2 \qquad \Gamma_1; \Delta \left|\frac{q-K^{\mathrm{con}}}{p}\right._{\mathbb{Q}} e \uparrow_b \exists \bar{z}{:}\bar{T}_z.\{\mathsf{Bool} \mid \psi\} \\ \Gamma_2; \Delta, \bar{z}{:}\bar{T}_z, [\top/\nu]\psi \left|\frac{p-K_1^{\mathrm{conT}}}{q'+K_2^{\mathrm{conT}}}\right._{\mathbb{Q}} t_1 \downarrow_b T \qquad \Gamma_2; \Delta, \bar{z}{:}\bar{T}_z, [\bot/\nu]\psi \left|\frac{p-K_1^{\mathrm{conF}}}{q'+K_2^{\mathrm{conF}}}\right._{\mathbb{Q}} t_2 \downarrow_b T\end{array}}{\Gamma; \Delta \left|\frac{q}{q'}\right._{\mathbb{Q}} \mathsf{if}\ e\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2 \downarrow_b T} \text{(BD:Cond)}$$

$$\frac{\begin{array}{c}\Gamma \mid\mid \Gamma_1; \Gamma_2 \\ \Gamma_1; \Delta \left|\frac{q-K^{\mathrm{mat}}}{r}\right._{\mathbb{Q}} e \uparrow_b \exists \bar{z}{:}\bar{T}_z.\{L^p(T') \mid \psi\} \qquad \Gamma_2; \Delta, \bar{z}{:}\bar{T}_z, [x'/\nu]g_{\mathtt{nil}}, [x'/\nu]\psi \left|\frac{r-K_1^{\mathrm{matN}}}{q'+K_2^{\mathrm{matN}}}\right._{\mathbb{Q}} t_1 \downarrow_b T \\ \Delta' = \Delta, \bar{z}{:}\bar{T}_z, x_1 : T', x_2 : \{L^p(T') \mid \psi_{\mathtt{cons}}\}, [x'/\nu]g_{\mathtt{cons}}, [x'/\nu]\psi \\ \Gamma_2, x_1 : T', x_2 : \{L^p(T') \mid \psi_{\mathtt{cons}}\}; \Delta' \left|\frac{r+p-K_1^{\mathrm{matC}}}{q'+K_2^{\mathrm{matC}}}\right._{\mathbb{Q}} t_2 \downarrow_b T\end{array}}{\Gamma; \Delta \left|\frac{q}{q'}\right._{\mathbb{Q}} \mathtt{match}(e; t_1; x_1, x_2.t_2) \downarrow_b T} \text{(BD:Match)}$$

$$\frac{f{:}T_x \xrightarrow{p/p'} T, x{:}T; \Delta, x{:}|T| \left|\frac{p}{p'}\right._{\mathbb{Q}} e \downarrow_b T}{\emptyset; \Delta \left|\frac{q+K^{\mathrm{fix}}}{q}\right._{\mathbb{Q}} \mathsf{fix}\ f.\lambda x.e \downarrow_b T_x \xrightarrow{p/p'} T} \text{(BD:Fix)}$$

Figure 13: Bidirectional Type Checking Rules (1 of 2)

$$\frac{\Gamma; \Delta \left\lfloor \frac{q}{q'} \right\rfloor_{\mathbb{Q}} e \uparrow_b T' \qquad \Delta \vdash T' <: T}{\Gamma; \Delta \left\lfloor \frac{q}{q'} \right\rfloor_{\mathbb{Q}} e \downarrow_b T} \text{ (BD:IE)} \quad \frac{\Gamma; \Delta \left\lfloor \frac{p}{p'} \right\rfloor_{\mathbb{Q}} t \downarrow_b T \qquad q \geq p \qquad q - p \geq q' - p'}{\Gamma; \Delta \left\lfloor \frac{q}{q'} \right\rfloor_{\mathbb{Q}} t \downarrow_b T} \text{ (BD:Relax)}$$

$$\frac{\Gamma; \Delta \left\lfloor \frac{q}{q'} \right\rfloor_{\mathbb{Q}} e \uparrow_b T}{\Gamma, x{:}T'; \Delta, x{:}|T'| \left\lfloor \frac{q}{q'} \right\rfloor_{\mathbb{Q}} e \uparrow_b T} \text{ (BD:WeakInf)} \quad \frac{\Gamma; \Delta \left\lfloor \frac{q}{q'} \right\rfloor_{\mathbb{Q}} t \downarrow_b T}{\Gamma, x{:}T'; \Delta, x{:}|T'| \left\lfloor \frac{q}{q'} \right\rfloor_{\mathbb{Q}} t \downarrow_b T} \text{ (BD:WeakChk)}$$

Figure 13: Bidirectional Type Checking Rules (2 of 2)

Lemma 3. *If $\emptyset; \Delta \vdash t_1 : T$ where T is not a scalar type and $\Gamma, x{:}T; \Delta, x{:}T \vdash t_2 : T'$ then it follows that $\Gamma; \Delta, x{:}T \vdash [t_1/x]t_2 : T'$.*

Proof. We go by induction on the derivation of t_2. In the LT:Var case, the claim holds trivially. Otherwise, we apply the induction hypothesis (weakening Δ sufficiently for t_1 so that it is of the proper shape) and reform the expression in question out of the post-substitution subterms. □

3.2.1 Language Correctness

In this section, we show that if a well-typed term t evaluates to a value v in a well-formed environment, then that value checks against type T, thereby satisfying the refinement specified by T.

Theorem 1. *If $\Delta \vdash_{\mathbb{Q}} V : \Gamma$, $V \vdash t \Downarrow v \mid (q, q')$, and $\Gamma; \Delta \vdash_{\mathbb{Q}} t : T$ then $\Delta \vdash_{\mathbb{Q}} v : T$.*

Proof. We go by induction on the evaluation and typing judgements, where the former takes priority.

(LT:Var-Base): If the type derivation ends with LT:Var-Base, then the derivation of the evaluation judgement ends with an application of E:Var. We have that $\Gamma = \Gamma', x{:}\{B \mid \psi\}$ and by $\Delta, x{:}\{|B| \mid \psi\} \vdash_{\mathbb{Q}} V : \Gamma$ that $\Delta \vdash_{\mathbb{Q}} V(x) : \{B \mid \psi\}$. Thus, by V:Self, we have that $\Delta, x{:}\{B \mid \psi\} \vdash_{\mathbb{Q}} V(x) : \{B \mid \nu = x\}$ as desired.

(LT:Var): If the type derivation ends with LT:Var, then the derivation of the evaluation judgement ends with an application of E:Var. We have that $\Gamma = \Gamma', x{:}T$ and by $\Delta, x{:}|T| \vdash_{\mathbb{Q}} V : \Gamma$ that $\Delta \vdash_{\mathbb{Q}} V(x) : T$. Therefore, since $v = V(x)$, $\Delta, x{:}|T| \vdash_{\mathbb{Q}} v : T$ as desired.

(LT:Fun): Suppose $t = \lambda x.t'$ and the type derivation ends with the rule LT:Fun. Then the derivation of the evaluation judgement ends with the rule E:Fun. By assumption we have that $\Delta \vdash_{\mathbb{Q}} V : \Gamma$, and by inversion on LT:Fun we have that $\Gamma, x : T_x; \Delta \vdash_{\mathbb{Q}} t' : T$. Therefore, by V:Lam, we get that $\Delta \vdash_{\mathbb{Q}} (\lambda x.t', V) : T_x \to T$ as desired.

$$\frac{\Delta \vdash \{|B| \mid \psi\} <: T}{\Gamma, x{:}\{B \mid \psi\}; \Delta, x{:}\{|B| \mid \psi\} \left|\frac{q+K^{\mathrm{var}}}{q}\right._{\mathbb{Q}} x \downarrow T \uparrow \{B \mid \nu = x\}} \text{ (RT:Var-Base)}$$

$$\frac{T \text{ is not a base type} \qquad \Delta \vdash T <: T'}{\Gamma, x{:}T; \Delta \left|\frac{q+K^{\mathrm{var}}}{q}\right._{\mathbb{Q}} x \downarrow T' \uparrow T} \text{ (RT:Var)}$$

$$\frac{\Delta \vdash \{\mathsf{Bool} \mid \nu = \top\} <: T}{\emptyset; \Delta \left|\frac{q+K^{\mathrm{bool}}}{q}\right._{\mathbb{Q}} \mathtt{true} \downarrow T \uparrow \{\mathsf{Bool} \mid \nu = \top\}} \text{ (RT:True)}$$

$$\frac{\Delta \vdash \{\mathsf{Bool} \mid \nu = \mathtt{false}\} <: T}{\emptyset; \Delta \left|\frac{q+K^{\mathrm{bool}}}{q}\right._{\mathbb{Q}} \mathtt{false} \downarrow T \uparrow \{\mathsf{Bool} \mid \nu = \mathtt{false}\}} \text{ (RT:False)}$$

$$\frac{\Delta \vdash \{\mathsf{Int} \mid \nu = n\} <: T}{\emptyset; \Delta \left|\frac{q+K^{\mathrm{int}}}{q}\right._{\mathbb{Q}} n \downarrow T \uparrow \{\mathsf{Bool} \mid \nu = n\}} \text{ (RT:Int)} \qquad \frac{\Delta \vdash \{L(T') \mid g_{\mathtt{nil}}\} <: T}{\emptyset; \Delta \left|\frac{q+K^{\mathrm{nil}}}{q}\right._{\mathbb{Q}} \mathtt{nil} \downarrow T \uparrow \{L(T') \mid g_{\mathtt{nil}}\}} \text{ (RT:Nil)}$$

$$\frac{\begin{array}{c}\Gamma \mid\mid \Gamma_1; \Gamma_2 \qquad \Gamma_1; \Delta \left|\frac{q-K_1^{\mathrm{cons}}}{p}\right._{\mathbb{Q}} e_1 \downarrow \mathsf{top} \uparrow T' \\ \Gamma_2; \Delta, x_1{:}|T'| \left|\frac{p-K_2^{\mathrm{cons}}}{q'+r+K_3^{\mathrm{cons}}}\right._{\mathbb{Q}} e_2 \downarrow \{L^r(T') \mid \psi_{\mathtt{cons}}\} \uparrow T'' \\ \Delta, x_1{:}|T'|, x_2{:}|T''| \vdash \{L^r(T') \mid g_{\mathtt{cons}}\} <: T\end{array}}{\Gamma; \Delta \left|\frac{q}{q'}\right._{\mathbb{Q}} \mathtt{cons}(e_1, e_2) \downarrow T \uparrow \exists x_1{:}|T'|, x_2{:}|T''|.\{L^r(T') \mid g_{\mathtt{cons}}\}} \text{ (RT:Cons)}$$

$$\frac{\Gamma, y{:}T_x; \Delta, y{:}|T_x| \left|\frac{p}{p'}\right._{\mathbb{Q}} t \downarrow [y/x]T \qquad \Gamma \Downarrow}{\Gamma; \Delta \left|\frac{q+K^{\mathrm{fun}}}{q}\right._{\mathbb{Q}} \lambda y.t \downarrow (x : T_x \xrightarrow{p/p'} T)} \text{ (RT:Fun)}$$

$$\frac{\begin{array}{c}\Gamma \mid\mid \Gamma_1; \Gamma_2 \\ \Gamma_1; \Delta \left|\frac{q-K_1^{\mathrm{app}}}{p}\right._{\mathbb{Q}} e_1 \downarrow \{B \mid \bot\} \xrightarrow{r-K_3^{\mathrm{app}}/q'+K_4^{\mathrm{app}}} T \uparrow x{:}\{B \mid \psi\} \xrightarrow{r-K_3^{\mathrm{app}}/q'+K_4^{\mathrm{app}}} T' \\ \Gamma_2; \Delta \left|\frac{p-K_2^{\mathrm{app}}}{r}\right._{\mathbb{Q}} e_2 \downarrow \{B \mid \psi\} \uparrow T_x \qquad \Delta, x{:}|T_x| \vdash T' <: T\end{array}}{\Gamma; \Delta \left|\frac{q}{q'}\right._{\mathbb{Q}} e_1\ e_2 \downarrow T \uparrow \exists x{:}|T_x|.T'} \text{ (RT:AppFO)}$$

$$\frac{\begin{array}{c}\Gamma \mid\mid \Gamma_1; \Gamma_2 \qquad \Gamma_1; \Delta \left|\frac{q-K_1^{\mathrm{app}}}{p}\right._{\mathbb{Q}} e \downarrow \mathsf{bot} \xrightarrow{r-K_3^{\mathrm{app}}/q'+K_4^{\mathrm{app}}} T \uparrow T'_x \xrightarrow{r-K_3^{\mathrm{app}}/q'+K_4^{\mathrm{app}}} T' \\ \Gamma_2; \Delta \left|\frac{p-K_2^{\mathrm{app}}}{r}\right._{\mathbb{Q}} f \downarrow T'_x\end{array}}{\Gamma; \Delta \left|\frac{q}{q'}\right._{\mathbb{Q}} e\ f \downarrow T \uparrow T'} \text{ (RT:AppHO)}$$

$$\frac{\begin{array}{c}\Gamma \mid\mid \Gamma_1; \Gamma_2 \qquad \Gamma_1; \Delta \left|\frac{q-K^{\mathrm{con}}}{p}\right._{\mathbb{Q}} e \downarrow \{\mathsf{Bool} \mid \top\} \uparrow \exists \bar{z}{:}\bar{T}_z.\{\mathsf{Bool} \mid \psi\} \\ \Gamma_2; \Delta, \bar{z}{:}\bar{T}_z, [\top/\nu]\psi \left|\frac{p-K_1^{\mathrm{conT}}}{q'+K_2^{\mathrm{conT}}}\right._{\mathbb{Q}} t_1 \downarrow T \qquad \Gamma_2; \Delta, \bar{z}{:}\bar{T}_z, [\bot/\nu]\psi \left|\frac{p-K_1^{\mathrm{conF}}}{q'+K_2^{\mathrm{conF}}}\right._{\mathbb{Q}} t_2 \downarrow T\end{array}}{\Gamma; \Delta \left|\frac{q}{q'}\right._{\mathbb{Q}} \mathsf{if}\ e\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2 \downarrow T} \text{ (RT:Cond)}$$

Figure 14: Round-trip Type Checking Rules (1 of 2)

$$\frac{\begin{array}{c}\Gamma \mid\mid \Gamma_1;\Gamma_2 \qquad \Gamma_1;\Delta \left|\frac{q-K^{\mathrm{mat}}}{r}\right._{\mathbb{Q}} e \downarrow \mathsf{top} \uparrow \exists \bar{z}{:}\bar{T}_z.\{L^p(T') \mid \psi\} \\ \Gamma_2;\Delta,\bar{z}{:}\bar{T}_z,[x'/\nu]g_{\mathtt{nil}},[x'/\nu]\psi \left|\frac{r-K_1^{\mathrm{matN}}}{q'+K_2^{\mathrm{matN}}}\right._{\mathbb{Q}} t_1 \downarrow T \\ \Delta' = \Delta,\bar{z}{:}\bar{T}_z,x_1:T',x_2:\{L^p(T') \mid \psi_{\mathtt{cons}}\},[x'/\nu]g_{\mathtt{cons}},[x'/\nu]\psi \\ \Gamma_2,x_1:T',x_2:\{L^p(T') \mid \psi_{\mathtt{cons}}\};\Delta' \left|\frac{r+p-K_1^{\mathrm{matC}}}{q'+K_2^{\mathrm{matC}}}\right._{\mathbb{Q}} t_2 \downarrow T\end{array}}{\Gamma;\Delta \left|\frac{q}{q'}\right._{\mathbb{Q}} \mathtt{match}(e;t_1;x_1,x_2.t_2) \downarrow T}\ (\text{RT:Match})$$

$$\frac{f{:}T_x \xrightarrow{p/p'} T, x{:}T;\Delta,x{:}|T| \left|\frac{p}{p'}\right._{\mathbb{Q}} e \downarrow T}{\emptyset;\Delta \left|\frac{q+K^{\mathrm{fix}}}{q}\right._{\mathbb{Q}} \mathsf{fix}\ f.\lambda x.e \downarrow T_x \xrightarrow{p/p'} T}\ (\text{RT:Fix})$$

$$\frac{\Gamma;\Delta \left|\frac{q}{q'}\right._{\mathbb{Q}} e \downarrow T \uparrow T'}{\Gamma;\Delta \left|\frac{q}{q'}\right._{\mathbb{Q}} e \downarrow T}\ (\text{RT:IE}) \qquad \frac{\Gamma;\Delta \left|\frac{p}{p'}\right._{\mathbb{Q}} t \downarrow T \qquad q \geq p \qquad q-p \geq q'-p'}{\Gamma;\Delta \left|\frac{q}{q'}\right._{\mathbb{Q}} t \downarrow T}\ (\text{RT:Relax})$$

$$\frac{\Gamma;\Delta \left|\frac{q}{q'}\right._{\mathbb{Q}} e \downarrow T \uparrow T'}{\Gamma,x{:}T'';\Delta,x{:}|T''| \left|\frac{q}{q'}\right._{\mathbb{Q}} e \downarrow T \uparrow T'}\ (\text{RT:WeakInf}) \qquad \frac{\Gamma;\Delta \left|\frac{q}{q'}\right._{\mathbb{Q}} t \downarrow T}{\Gamma,x{:}T';\Delta,x{:}|T'| \left|\frac{q}{q'}\right._{\mathbb{Q}} t \downarrow T}\ (\text{RT:WeakChk})$$

Figure 14: Round-trip Type Checking Rules (2 of 2)

(LT:App): In this case, we have $t = e\ t'$ and therefore we have that the final rule of the evaluation judgement is E:App. Therefore, by inversion on the typing judgement, we get that

$$\Gamma_1;\Delta \vdash_{\mathbb{Q}} e : T_x \to T \tag{3.1}$$

$$\Gamma_2;\Delta \vdash_{\mathbb{Q}} t' : T_x \tag{3.2}$$

By inversion on the evaluation judgement, we get that

$$V \vdash e \Downarrow (\lambda x.t'', V') \mid (q, q') \tag{3.3}$$

$$V \vdash t' \Downarrow v \mid (p, p') \tag{3.4}$$

$$V'[x \mapsto v] \vdash t'' \Downarrow v' \mid (r, r') \tag{3.5}$$

By Lemma 2, $\Delta \vdash_{\mathbb{Q}} V : \Gamma$ gives us that $\Delta \vdash_{\mathbb{Q}} V : \Gamma_1$ and $\Delta \vdash_{\mathbb{Q}} V : \Gamma_2$ hold, so we apply the I.H. to $\Delta \vdash_{\mathbb{Q}} V : \Gamma_1$, (3.1), and (3.3) to get that $\Delta \vdash_{\mathbb{Q}} (\lambda x.t'', V') : T_x \to T$ holds. Then, we apply the I.H. to $\Delta \vdash_{\mathbb{Q}} V : \Gamma_2$, (3.2), and (3.4) to get that $\Delta \vdash_{\mathbb{Q}} v : T_x$ holds. By inversion on V:Lam, we have that there exists some Γ' such that $\Delta \vdash_{\mathbb{Q}} V' : \Gamma'$ and

$$\Gamma', x : T_x; \Delta, x : T_x \vdash_{\mathbb{Q}} t'' : T \tag{3.6}$$

Moreover, by the conclusions above, we get that

$$\Delta, x{:}T_x \vdash_{\mathbb{Q}} V'[x \mapsto v] : \Gamma', x : T_x \tag{3.7}$$

We can apply the I.H. again to (3.5), (3.6), and (3.7) and get that $\Delta \vdash_{\mathbb{Q}} v' : T$ as desired.

(LT:Cond): Here, the evaluation ends with an application of either E:CondT or E:CondF; assume it ends with E:CondT, as the other case is similar.

By Lemma 2 and $\Delta \vdash_{\mathbb{Q}} V : \Gamma$, we get that $\Delta \vdash_{\mathbb{Q}} V : \Gamma_2$. Furthermore, by repeatedly using VC:DExt and VC:BExt it follows that $\Delta, \bar{z}{:}\bar{T}_z, [\top/\nu]\psi \vdash_{\mathbb{Q}} V : \Gamma_2$. We thus use $\Gamma_2; \Delta, \bar{z}{:}\bar{T}_z, [\top/\nu]\psi \vdash_{\mathbb{Q}} t_1 : T$ to apply the induction hypothesis to the second premise $V \vdash t_1 \Downarrow v \mid (p, p')$ of E:CondT, thereby getting that $\Delta, \bar{z}{:}\bar{T}_z, [\top/\nu]\psi \vdash_{\mathbb{Q}} v : T$ as desired.

(LT:Fix): The evaluation ends with an application of E:Fix. We thus have

$$f{:}T_x \to T, x{:}T; \Delta, x{:}T \vdash_{\mathbb{Q}} t' : T$$

as a premise from LT:Fix. Additionally, by assumption, we have

$$\emptyset; \Delta \vdash_{\mathbb{Q}} \mathsf{fix}\ f.\lambda x.t' : T_x \to T$$

Therefore, we can apply Lemma 3 and get that

$$x{:}T; \Delta, x{:}T \vdash_{\mathbb{Q}} [\mathsf{fix}\ f.\lambda x.t'/f]t' : T$$

Since we have by assumption that $\Delta \vdash_{\mathbb{Q}} V : \emptyset$, we get by V:Lam that $\Delta \vdash_{\mathbb{Q}} (\lambda x.t'', V) : T_x \to T$ as desired, where $t'' = [\mathsf{fix}\ f.\lambda x.t'/f]t'$.

(LT:Nil): The evaluation ends with an application of E:Nil. We thus get that $\Delta \vdash_{\mathbb{Q}} \mathtt{nil} : \{L(T) \mid g_{\mathtt{nil}}\}$ by V:Nil as desired.

(LT:True), (LT:False), (LT:Int): Similar to the (LT:Nil) case.

(LT:Cons): In this case, we have $t = \mathtt{cons}(e_1, e_2)$ and therefore we have that the final rule of the evaluation judgement is E:Cons. Therefore, by inversion on the typing judgement, we get that

$$\Gamma_1; \Delta \left|^{\frac{q-K_1^{\mathrm{cons}}}{p}}_{\mathbb{Q}}\ e_1 : T \right. \tag{3.8}$$

$$\Gamma_2; \Delta, x_1{:}|T| \left|^{\frac{p-K_2^{\mathrm{cons}}}{q'+r+K_3^{\mathrm{cons}}}}_{\mathbb{Q}}\ e_2 : \{L^r(T) \mid \psi_{\mathtt{cons}}\} \right. \tag{3.9}$$

Furthermore, by inversion on the evaluation judgement, we get that

$$V \vdash e_1 \Downarrow v_1 \mid (q, q') \tag{3.10}$$

$$V \vdash e_2 \Downarrow v_2 \mid (p, p') \tag{3.11}$$

By Lemma 2 and $\Delta \vdash_{\mathbb{Q}} V : \Gamma$ we get that $\Delta \vdash_{\mathbb{Q}} V : \Gamma_1$ and $\Delta \vdash_{\mathbb{Q}} V : \Gamma_2$. Therefore, we apply the I.H. to each and get that $\Delta \vdash_{\mathbb{Q}} v_1 : T$ and $\Delta, x{:}|T| \vdash_{\mathbb{Q}} v_2 : \{L^r(T) \mid \psi_{\texttt{cons}}\}$. Using V:Cons we can derive that

$$\Delta \vdash_{\mathbb{Q}} \texttt{cons}(v_1, v_2) : \{L^q(T') \mid [\overline{v_1}, \overline{v_2}/x_1, x_2]g_{\texttt{cons}}\}$$

and by applying V:Wit twice, we get that

$$\Delta \vdash_{\mathbb{Q}} \texttt{cons}(v_1, v_2) : \exists x_1{:}|T|, x_2{:}\{L^r(|T|) \mid \psi_{\texttt{cons}}\}.\{L^q(T') \mid g_{\texttt{cons}}\}$$

as desired.

(LT:Match): The evaluation ends with an application of E:MatchN or E:MatchC. By Lemma 2 we have that $\Delta \vdash_{\mathbb{Q}} V : \Gamma_1$ and $\Delta \vdash_{\mathbb{Q}} V : \Gamma_2$. We thus apply the I.H. to the first premise of both evaluation rules and either get that $\Delta \vdash_{\mathbb{Q}} \texttt{nil} : \exists \bar{z}{:}\overline{T_z}.\{L(T') \mid \psi\}$ or $\Delta \vdash_{\mathbb{Q}} \texttt{cons}(v_1, v_2) : \exists \bar{z}{:}\overline{T_z}.\{L(T') \mid \psi\}$ for some v_1, v_2. The former case is similar to the case (LT:Cond). In the latter case, we have by several steps of appropriate inversion (on V:Cons and on V:Wit and V:Self as necessary) that $\Delta \vdash_{\mathbb{Q}} v_1 : T'$ and $\Delta, x_1{:}|T'| \vdash_{\mathbb{Q}} v_2 : \{L(T') \mid \psi_{\texttt{cons}}\}$. Therefore, by applying VC:DExt several times we have that $\Delta, \bar{z}{:}\overline{T_z} \vdash_{\mathbb{Q}} V : \Gamma_2$, and by applying VC:Ext twice, we get that

$$\Delta, \bar{z}{:}\overline{T_z}, x_1 : |T'|, x_2 : \{L(|T'|) \mid \psi_{\texttt{cons}}\} \vdash_{\mathbb{Q}} V[x_1 \mapsto v_1, x_2 \mapsto v_2] : \Gamma_2, x_1 : T', x_2 : \{L(T') \mid \psi_{\texttt{cons}}\}$$

Finally, we can apply VC:BExt several times and get that

$$\Delta' \vdash_{\mathbb{Q}} V[x_1 \mapsto v_1, x_2 \mapsto v_2] : \Gamma_2, x_1 : T', x_2 : \{L(T') \mid \psi_{\texttt{cons}}\}$$

where

$$\Delta' = \Delta, \bar{z}{:}\overline{T_z}, x_1 : |T'|, x_2 : \{L(|T'|) \mid \psi_{\texttt{cons}}\}, [x'/\nu]g_{\texttt{cons}}, [x'/\nu]\psi$$

thus allowing us to apply the I.H. to the second premise of E:MatchC and LT:Match to get $\Delta \vdash_{\mathbb{Q}} v : T$.

(LT:Sub): Assume the derivation of the typing judgement ends with an application of LT:Sub. Then we have that $\Gamma; \Delta \vdash_{\mathbb{Q}} t : T$, $\Delta \vdash T <: T'$, and $\Delta \vdash_{\mathbb{Q}} T'$. Since we have by assumption that $\Delta \vdash_{\mathbb{Q}} V : \Gamma$ and $V \vdash t \Downarrow v \mid (q, q')$, we can apply the I.H. and get that $\Delta \vdash_{\mathbb{Q}} v : T$. Finally, since all inhabitants of T are inhabitants of T', we get that $\Delta \vdash_{\mathbb{Q}} v : T'$.

(LT:Weaken): If the derivation of the typing judgement ends with an application of LT:Weaken then we have $\Gamma', x : T'; \Delta, x{:}|T'| \vdash^{\frac{q}{q'}}_{\mathbb{Q}} t : T$ for some Γ'. By assumption we have that $\Delta \vdash_{\mathbb{Q}} V : \Gamma', x : T'$, and therefore that $\Delta \vdash_{\mathbb{Q}} V : \Gamma'$. We can thus apply the induction hypothesis and get that $\Delta \vdash_{\mathbb{Q}} v : T$; by V:Weak, we get that $\Delta, x{:}|T'| \vdash_{\mathbb{Q}} v : T$ as desired.

□

3.2.2 Soundness of Resource Analysis

Here, we show that if a well-typed term t evaluates to a value v in a well-formed environment, then the initial potential of the context gives an upper bound on the watermark of the actual resource usage. Furthermore, we claim that the difference between the initial and final potential gives an upper bound on the actual resource consumption.

Theorem 2. *Let* $\Delta \vdash_{\mathbb{Q}} V : \Gamma$ *and* $\Gamma; \Delta \left|\frac{q}{q'}_{\mathbb{Q}}\right. t : T$.

If $V \vdash t \Downarrow v \mid (p, p')$ *then* $p \leq \Phi(V : \Gamma) + q$ *and* $p - p' \leq \Phi(V : \Gamma) + q - (\Phi(v : T) + q')$.

Proof. We go by induction on the derivations of the evaluation and type judgements, where the former takes priority.

(LT:Weaken): If the derivation of the typing judgement ends with an application of LT:Weaken then we have $\Gamma', x : T'; \Delta \left|\frac{q}{q'}_{\mathbb{Q}}\right. t : T$ for some Γ'. By assumption we have that $\Delta \vdash_{\mathbb{Q}} V : \Gamma', x : T'$, and therefore that $\Delta \vdash_{\mathbb{Q}} V : \Gamma'$. We can thus apply the induction hypothesis and get that $p \leq \Phi(V : \Gamma') + q$ and $p - p' \leq \Phi(V : \Gamma') + q - (\Phi(v : T) + q')$. Since $q, q' \in \mathbb{Q}_0^+$, by the definition of Φ we get that $\Phi(V : \Gamma') \leq \Phi(V : \Gamma)$ and thus the claim holds.

(LT:Sub): Assume the derivation of the typing judgement ends with an application of LT:Sub. Then we have that $\Gamma; \Delta \vdash_{\mathbb{Q}} t : T$, $\Delta \vdash T <: T'$, and $\Delta \vdash_{\mathbb{Q}} T'$. Since we have by assumption that $\Delta \vdash_{\mathbb{Q}} V : \Gamma$ and $V \vdash t \Downarrow v \mid (q, q')$, we can apply the I.H. and get that $p \leq \Phi(V{:}\Gamma) + q$ and $p - p' \leq \Phi(V{:}\Gamma) + q - (\Phi(v{:}T) + q')$. Moreover, by Theorem 1, we have that $\Delta \vdash_{\mathbb{Q}} v : T$, and since all inhabitants of T are inhabitants of T', we get that $\Delta \vdash_{\mathbb{Q}} v : T'$. By Lemma 1 it follows that $\Phi(v{:}T) \geq \Phi(v{:}T')$. Therefore, it follows that that $p - p' \leq \Phi(V{:}\Gamma) + q - (\Phi(v{:}T') + q')$ and the claim holds.

(LT:Relax): We apply the I.H. to the evaluation judgement and to the premise $\Gamma; \Delta \left|\frac{r}{r'}_{\mathbb{Q}}\right. t : T$ of LT:Relax. Then we have $p \leq \Phi(V : \Gamma) + r$ and $p - p' \leq \Phi(V : \Gamma) + q - (\Phi(v : T) + q')$. From the premises of LT:Relax we have that $q \geq r$ and $q - r \geq q' - r'$ and thus $q - q' \geq r - r'$, from which the claim follows.

(LT:Var): Let t be a variable x that has been evaluated with the rule E:Var. First, assume that $K^{\text{var}} \geq 0$. Then by definition $p = K^{\text{var}}$ and $p' = 0$. The type judgement has been derived by a single application of the rule LT:Var. Thus we have $0 \leq q' = q - K^{\text{var}}$ and therefore $p = K^{\text{var}} \leq q \leq \Phi(V(x) : T) + q$. Furthermore, it follows by E:Var that $v = V(x)$ and thus $p - p' = K^{\text{var}} = q - q' = \Phi(V(x) : T) + q - (\Phi(v : T) + q')$. Assume now that $K^{\text{var}} < 0$. Then it follow by definition that $p = 0$ and $p' = -K^{\text{var}}$. Thus $p = 0 \leq \Phi(V(x) : T) + q$. We have again that $q - q' = K^{\text{var}} = p - p'$, and so the second part of the claim follows as above.

(LT:Var-Base): Same as the (LT:Var) case, since refinements do not affect the potential.

(LT:Nil): If the type derivation ends with an application of LT:Nil then we have $t = \texttt{nil}$, $T = \{L^r(T') \mid g_{\texttt{nil}}\}$ for some T', and $0 \leq q' = q - K^{\text{nil}}$. From the evaluation judgement we get that $v = \texttt{nil}$ as well. If $K^{\text{nil}} \geq 0$ then $p = K^{\text{nil}}$ and $p' = 0$. Thus $p = K^{\text{nil}} \leq q = \Phi(V : \emptyset) + q - (\Phi(\texttt{nil} : \{L^r(T') \mid g_{\texttt{nil}}\}) + q')$. Otherwise if $K^{\text{nil}} < 0$ then $p = 0$ and $p' = -K^{\text{nil}}$. Then $p \leq q$ and again $p - p' = K^{\text{nil}}$.

(LT:True), (LT:False), (LT:Int): Similar to the case (LT:Nil).

(LT:Cons): Let t be a constructor application of the form $\texttt{cons}(e_1, e_2)$. The evaluation of t thus ends with an application of E:Cons. As a result, we have $V \vdash e_1 \Downarrow v_1 \mid (r_1, r_1')$ and $V \vdash e_2 \Downarrow v_2 \mid (r_2, r_2')$ with

$$(p, p') = K_1^{\text{cons}} \cdot (r_1, r_1') \cdot K_2^{\text{cons}} \cdot (r_2, r_2') \cdot K_3^{\text{cons}} \tag{3.12}$$

The derivation of the type judgement for e ends with an application of LT:Cons. Therefore, $\Gamma \mid\mid \Gamma_1; \Gamma_2$, $\Gamma_1; \Delta \left|\frac{s_1}{s_1'}_{\mathbb{Q}}\right. e_1 : T$, $\Gamma_2; \Delta, x_1{:}|T| \left|\frac{s_2}{s_2'}_{\mathbb{Q}}\right. e_2 : \{L^s(T) \mid \psi_{\texttt{cons}}\}$, and

$$q = s_1 + K_1^{\text{cons}} \tag{3.13}$$

$$s_1' = s_2 + K_2^{\text{cons}} \tag{3.14}$$

$$q' = s_2' - s - K_3^{\text{cons}} \tag{3.15}$$

By the definition of Φ we get that

$$\Phi(V{:}\Gamma) = \Phi(V{:}\Gamma_1) + \Phi(V{:}\Gamma_2) \tag{3.16}$$

By Lemma 2 and the assumption $\Delta \vdash_{\mathbb{Q}} V : \Gamma$ we also have $\Delta \vdash_{\mathbb{Q}} V : \Gamma_1$ and can apply the I.H. to the evaluation judgement for e_1 to derive

$$r_1 \leq \Phi(V{:}\Gamma_1) + s_1 \tag{3.17}$$

$$r_1 - r_1' \leq \Phi(V{:}\Gamma_1) + s_1 - (\Phi(v_1{:}T) + s_1') \tag{3.18}$$

As above, it follows that $\Delta \vdash_{\mathbb{Q}} V : \Gamma_2$ from Lemma 2 and $\Delta \vdash_{\mathbb{Q}} V : \Gamma$ and thus again by induction

$$r_2 \leq \Phi(V{:}\Gamma_2) + s_2 \tag{3.19}$$

$$r_2 - r_2' \leq \Phi(V{:}\Gamma_2) + s_2 - (\Phi(v_2{:}\{L^s(T) \mid \psi_{\texttt{cons}}\}) + s_2') \tag{3.20}$$

Now let

$$\begin{aligned}(u, u') \quad = \quad & K_1^{\text{cons}} \cdot (\Phi(V{:}\Gamma_1) + s_1, \Phi(v_1{:}T) + s_1') \cdot K_2^{\text{cons}} \cdot \\ & (\Phi(V{:}\Gamma_2) + s_2, \Phi(v_2{:}\{L^s(T) \mid \psi_{\texttt{cons}}\}) + s_2') \cdot K_3^{\text{cons}}\end{aligned}$$

Then it follows that

$$\begin{aligned}(u,u') &\overset{(3.14,\ 3.15)}{=} K_1^{\text{cons}} \cdot (\Phi(V{:}\Gamma_1) + s_1, \Phi(v_1{:}T) + s_1' - K_2^{\text{cons}}) \cdot \\ &\quad (\Phi(V{:}\Gamma_2) + s_2, \Phi(v_2{:}\{L^s(T) \mid \psi_{\texttt{cons}}\}) + s_2' - K_3^{\text{cons}}) \\ &= K_1^{\text{cons}} \cdot (v + \Phi(V{:}\Gamma_1) + s_1, v')\end{aligned}$$

for $v, v' \in \mathbb{Q}_0^+$ with

$$\begin{aligned}v &\leq \Phi(V{:}\Gamma_2) + s_2 - (\Phi(v_1{:}T) + s_1' - K_2^{\text{cons}}) \\ &= \Phi(V{:}\Gamma_2) + s_2 - (s_1' - K_2^{\text{cons}}) \\ &\overset{(3.14)}{=} \Phi(V{:}\Gamma_2)\end{aligned}$$

and thus

$$\begin{aligned}u &\leq \max(0, \Phi(V{:}\Gamma_1) + \Phi(V{:}\Gamma_2) + s_1 + K_1^{\text{cons}}) \\ &\overset{(3.13)}{\leq} \Phi(V{:}\Gamma) + q\end{aligned}$$

Finally, it follows by applying Proposition 1 to (3.17), (3.19), and (3.12) that $u \geq p$.

For the second part of the statement we apply the properties of watermarks to (3.12) and derive the following.

$$\begin{aligned}p - p' &= r_1 - r_1' + r_2 - r_2' + K_1^{\text{cons}} + K_2^{\text{cons}} + K_3^{\text{cons}} \\ &\overset{(3.20,\ 3.18)}{\leq} \Phi(V{:}\Gamma_1) + s_1 - (\Phi(v_1{:}T) + s_1') + \Phi(V{:}\Gamma_2) \\ &\quad + s_2 - (\Phi(v_2{:}\{L^s(T) \mid \psi_{\texttt{cons}}\}) + s_2') + K_1^{\text{cons}} + K_2^{\text{cons}} + K_3^{\text{cons}} \\ &= (\Phi(V{:}\Gamma_1) + \Phi(V{:}\Gamma_2) + s_1) - \Phi(v_1{:}T) \\ &\quad + (s_2 + K_2^{\text{cons}} - s_1') - (\Phi(v_2{:}\{L^s(T) \mid \psi_{\texttt{cons}}\}) + s_2') + K_1^{\text{cons}} + K_3^{\text{cons}} \\ &\overset{(3.14)}{=} \Phi(V{:}\Gamma_1) + \Phi(V{:}\Gamma_2) + s_1 - (\Phi(v_1{:}T) + \Phi(v_2{:}\{L^s(T) \mid \psi_{\texttt{cons}}\}) + s_2') \\ &\quad + K_1^{\text{cons}} + K_3^{\text{cons}} \\ &= \Phi(V{:}\Gamma) + s_1 + K_1^{\text{cons}} - (\Phi(v_1{:}T) + \Phi(v_2{:}\{L^s(T) \mid \psi_{\texttt{cons}}\}) + s_2' - K_3^{\text{cons}}) \\ &\overset{(3.13,\ 3.15)}{\leq} \Phi(V{:}\Gamma) + q - (\Phi(v_1{:}T) + \Phi(v_2{:}\{L^s(T) \mid \psi_{\texttt{cons}}\}) + q' + s) \\ &= \Phi(V{:}\Gamma) + q - (\Phi(\texttt{cons}(v_1, v_2){:}L^s(T)) + q')\end{aligned}$$

(LT:Cond): Assume that e is a conditional branch of the form $\mathsf{if}\ e\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2$. In this case, e must have been evaluated with either E:CondT or E:CondF. The cases are similar, so we assume the former. Therefore, we have by inversion that $V \vdash e \Downarrow \texttt{true} \mid (r_1, r_1')$ and $V \vdash t_1 \Downarrow v \mid (r_2, r_2')$ for some r_1, r_1', r_2, r_2' with

$$(p, p') = K^{\text{con}} \cdot (r_1, r_1') \cdot K_1^{\text{conT}} \cdot (r_2, r_2') \cdot K_2^{\text{conT}} \tag{3.21}$$

Similarly, the typing judgement has been derived by an application of LT:Cond and thus $\Gamma \mid\mid \Gamma_1; \Gamma_2$, $\Gamma_1; \Delta \left|^{s_1}_{s_1'\ \mathbb{Q}}\right. e : \exists \bar{z}{:}\bar{T}_z.\{\mathsf{Bool} \mid \psi\}$, and $\Gamma_2; \Delta, \bar{z}{:}\bar{T}_z, [\top/\nu]\psi \left|^{s_2}_{s_2'\ \mathbb{Q}}\right. t_1 : T$. Furthermore,

$$q = s_1 + K^{\mathrm{con}} \tag{3.22}$$

$$s_1' = s_2 + K_1^{\mathrm{conT}} \tag{3.23}$$

$$q' = s_2' - K_2^{\mathrm{conT}} \tag{3.24}$$

for some s, s'. By Lemma 2 and our assumption $\Delta \vdash_{\mathbb{Q}} V : \Gamma$, we get that $\Delta \vdash_{\mathbb{Q}} V : \Gamma_1$, and we can thus apply the induction hypothesis and have

$$r_1 \leq \Phi(V : \Gamma_1) + s_1 \tag{3.25}$$

$$r_1 - r_1' \leq \Phi(V : \Gamma_1) + s_1 - (\Phi(\mathtt{true} : \exists \bar{z}{:}\bar{T}_z.\{\mathsf{Bool} \mid \psi\}) + s_1') \tag{3.26}$$

As above, it follows that $\Delta \vdash_{\mathbb{Q}} V : \Gamma_2$ from Lemma 2 and $\Delta \vdash_{\mathbb{Q}} V : \Gamma$. By repeatedly applying VC:DExt and VC:BExt we get that $\Delta, \bar{z}{:}\bar{T}_z, [\top/\nu]\psi \vdash_{\mathbb{Q}} V : \Gamma_2$. We can again apply our I.H. and get that

$$r_2 \leq \Phi(V : \Gamma_2) + s_2 \tag{3.27}$$

$$r_2 - r_2' \leq \Phi(V : \Gamma_2) + s_2 - (\Phi(v : T) + s_2') \tag{3.28}$$

Now let

$$\begin{aligned}(u, u') \quad = \quad & K^{\mathrm{con}} \cdot (\Phi(V : \Gamma_1) + s_1, \Phi(\mathtt{true} : \exists \bar{z}{:}\bar{T}_z.\{\mathsf{Bool} \mid \psi\}) + s_1') \cdot \\ & K_1^{\mathrm{conT}} \cdot (\Phi(V : \Gamma_2) + s_2, \Phi(v : T) + s_2') \cdot K_2^{\mathrm{conT}}\end{aligned}$$

Then it follows that

$$\begin{aligned}(u, u') \quad \overset{(3.23,\ 3.24)}{=} \quad & K^{\mathrm{con}} \cdot (\Phi(V : \Gamma_1) + s_1, \Phi(\mathtt{true} : \exists \bar{z}{:}\bar{T}_z.\{\mathsf{Bool} \mid \psi\}) + s_1' - K_1^{\mathrm{conT}}) \cdot \\ & (\Phi(V : \Gamma_2) + s_2, \Phi(v : T) + s_2' - K_2^{\mathrm{conT}}) \\ = \quad & K^{\mathrm{con}} \cdot (v + \Phi(V{:}\Gamma_1) + s_1, v')\end{aligned}$$

for $v, v' \in \mathbb{Q}_0^+$ with

$$\begin{aligned}v \quad \leq \quad & \Phi(V{:}\Gamma_2) + s_2 - (\Phi(\mathtt{true} : \exists \bar{z}{:}\bar{T}_z.\{\mathsf{Bool} \mid \psi\}) + s_1' - K_1^{\mathrm{conT}}) \\ = \quad & \Phi(V{:}\Gamma_2) + s_2 - (s_1' - K_1^{\mathrm{conT}}) \\ \overset{(3.23)}{=} \quad & \Phi(V{:}\Gamma_2)\end{aligned}$$

and thus

$$\begin{array}{rcl} u & \leq & \max(0, \Phi(V{:}\Gamma_1) + \Phi(V{:}\Gamma_2) + s_1 + K^{\mathrm{con}}) \\ & \overset{(3.22)}{\leq} & \Phi(V{:}\Gamma) + q \end{array}$$

Finally, it follows by applying Proposition 1 to (3.25), (3.27), and (3.21) that $u \geq p$.

For the second part of the statement we apply the properties of watermarks to (3.21) and derive the following.

$$\begin{array}{rcl} p - p' & = & r_1 - r_1' + r_2 - r_2' + K^{\mathrm{con}} + K_1^{\mathrm{conT}} + K_2^{\mathrm{conT}} \\ & \overset{(3.28,\ 3.26)}{\leq} & \Phi(V : \Gamma_1) + s_1 - (\Phi(\mathtt{true} : \exists \bar{z}{:}\bar{T}_z.\{\mathsf{Bool} \mid \psi\}) + s_1') \\ & & +\Phi(V : \Gamma_2) + s_2 - (\Phi(v : T) + s_2') + K^{\mathrm{con}} + K_1^{\mathrm{conT}} + K_2^{\mathrm{conT}} \\ & = & (\Phi(V{:}\Gamma_1) + \Phi(V{:}\Gamma_2) + s_1) - \Phi(\mathtt{true} : \exists \bar{z}{:}\bar{T}_z.\{\mathsf{Bool} \mid \psi\}) \\ & & +(s_2 + K_1^{\mathrm{cons}} - s_1') - (\Phi(v{:}T) + s_2') + K^{\mathrm{con}} + K_2^{\mathrm{cons}} \\ & \overset{(3.23)}{=} & \Phi(V{:}\Gamma_1) + \Phi(V{:}\Gamma_2) + s_1 - (\Phi(\mathtt{true} : \exists \bar{z}{:}\bar{T}_z.\{\mathsf{Bool} \mid \psi\}) \\ & & +\Phi(v{:}T) + s_2') + K^{\mathrm{con}} + K_2^{\mathrm{conT}} \\ & = & \Phi(V{:}\Gamma) + s_1 + K^{\mathrm{con}} - (\Phi(v{:}T) + s_2' - K_2^{\mathrm{conT}}) \\ & \overset{(3.22,\ 3.24)}{\leq} & \Phi(V{:}\Gamma) + q - (\Phi(v{:}T) + q') \end{array}$$

We note that the case where e evaluates to `false` is similar.

(LT:Fun): Let t be a function abstraction of the form $\lambda x.t'$. Therefore, the derivation of the evaluation judgement ends with an application of E:Fun and the derivation of the typing judgement ends with an application of LT:Fun. Furthermore, we have $T = x{:}T_x \xrightarrow{s/s'} T'$ for some T_x, T', s, s', and $0 \leq q' = q - K^{\mathrm{fun}}$. In addition, from the last premise of LT:Fun we have that $\Gamma \downdownarrows$ and thus $\Phi(V{:}\Gamma) = 0$. From the evaluation judgement we get that $v = (\lambda x.t', V)$ as well. If $K^{\mathrm{fun}} \geq 0$ then $p = K^{\mathrm{fun}}$ and $p' = 0$. Thus $p = K^{\mathrm{fun}} \leq q = \Phi(V : \Gamma) + q$. By the definition of Φ, we get that $\Phi((\lambda x.t', V) : T_x \xrightarrow{s/s'} T') = 0$. Thus $p - p' = K^{\mathrm{fun}} = \Phi(\Gamma) + q - (\Phi((\lambda x.t', V) : T_x \xrightarrow{s/s'} T') + q')$. If $K^{\mathrm{fun}} < 0$ then $p = 0$ and $p' = -K^{\mathrm{fun}}$. Then $p \leq q$ and again $p - p' = K^{\mathrm{fun}}$.

(LT:Fix): Similar to the case (LT:Fun).

(LT:App): Let t be a function application of the form $e\ t'$. The evaluation of t thus ends with an application of E:App. As a result, we have $V \vdash e \Downarrow (\lambda x.t'', V') \mid (r_1, r_1')$, $V \vdash t' \Downarrow v \mid (r_2, r_2')$, and $V'[x \mapsto v] \vdash t'' \Downarrow v' \mid (r_3, r_3')$ with

$$(p, p') = K_1^{\mathrm{app}} \cdot (r_1, r_1') \cdot K_2^{\mathrm{app}} \cdot (r_2, r_2') \cdot K_3^{\mathrm{app}} \cdot (r_3, r_3') \cdot K_4^{\mathrm{app}} \tag{3.29}$$

The derivation of the type judgement for e ends with an application of LT:App. Therefore, $\Gamma \mid\mid \Gamma_1; \Gamma_2$, $\Gamma_1; \Delta \left|^{s_1}_{s_1'\,\mathbb{Q}}\right. e : (x{:}T_x \xrightarrow{s_3/s_3'} T)$, $\Gamma_2; \Delta \left|^{s_2}_{s_2'\,\mathbb{Q}}\right. t' : T_x$, and

$$q = s_1 + K_1^{\text{app}} \tag{3.30}$$

$$s_1' = s_2 + K_2^{\text{app}} \tag{3.31}$$

$$s_2' = s_3 + K_3^{\text{app}} \tag{3.32}$$

$$q' = s_3' - K_4^{\text{app}} \tag{3.33}$$

By the definition of Φ we get that

$$\Phi(V{:}\Gamma) = \Phi(V{:}\Gamma_1) + \Phi(V{:}\Gamma_2) \tag{3.34}$$

By Lemma 2 and $\Delta \vdash_{\mathbb{Q}} V : \Gamma$ we also have $\Delta \vdash_{\mathbb{Q}} V : \Gamma_1$ and can apply the I.H. to the evaluation judgement for e_1 to derive

$$r_1 \leq \Phi(V{:}\Gamma_1) + s_1 \tag{3.35}$$

$$r_1 - r_1' \leq \Phi(V{:}\Gamma_1) + s_1 - (\Phi((\lambda x.t'', V'){:}(x{:}T_x \xrightarrow{s_3/s_3'} T)) + s_1') \tag{3.36}$$

As above, it follows that $\Delta \vdash_{\mathbb{Q}} V : \Gamma_2$ holds and thus again by induction

$$r_2 \leq \Phi(V{:}\Gamma_2) + s_2 \tag{3.37}$$

$$r_2 - r_2' \leq \Phi(V{:}\Gamma_2) + s_2 - (\Phi(v{:}T_x) + s_2') \tag{3.38}$$

By inversion on V:Lam with $(\lambda x.t'', V')$, we have that there exists some Γ' such that $\Delta \vdash_{\mathbb{Q}} V' : \Gamma'$, $\Gamma', x{:}T_x; \Delta \left|^{s_3}_{s_3'\,\mathbb{Q}}\right. t'' : T$, and $\Gamma' \downdownarrows$. By Theorem 1, we have that $\Delta \vdash_{\mathbb{Q}} v : T_x$, so by definition we have that $\Delta \vdash_{\mathbb{Q}} V'[x \mapsto v] : \Gamma', x{:}T_x$. We now apply the induction hypothesis once more to derive

$$r_3 \leq \Phi(V'[x \mapsto v] : \Gamma', x{:}T_x) + s_3 \tag{3.39}$$

$$r_3 - r_3' \leq \Phi(V'[x \mapsto v] : \Gamma', x{:}T_x) + s_3 - (\Phi(v'{:}T) + s_3') \tag{3.40}$$

Now let

$$\begin{aligned}(u, u') \quad = \quad & K_1^{\text{app}} \cdot (\Phi(V{:}\Gamma_1) + s_1, \Phi((\lambda x.t'', V'){:}(x{:}T_x \xrightarrow{s_3/s_3'} T)) + s_1') \cdot K_2^{\text{app}} \cdot \\ & (\Phi(V{:}\Gamma_2) + s_2, \Phi(v{:}T_x) + s_2') \cdot K_3^{\text{app}} \cdot \\ & (\Phi(V'[x \mapsto v] : \Gamma', x{:}T_x) + s_3, \Phi(v'{:}T) + s_3') \cdot K_4^{\text{app}}\end{aligned}$$

Then it follows that

$$
\begin{array}{rcl}
(u,u') & \overset{(3.31,\ 3.32,\ 3.33)}{=} & K_1^{\text{app}} \cdot (\Phi(V{:}\Gamma_1) + s_1, \Phi((\lambda x.t'', V'){:}(x{:}T_x \xrightarrow{s_3/s_3'} T)) + s_1' - K_2^{\text{app}}) \cdot \\
& & (\Phi(V{:}\Gamma_2) + s_2, \Phi(v{:}T_x) + s_2' - K_3^{\text{app}}) \cdot \\
& & (\Phi(V'[x \mapsto v] : \Gamma', x{:}T_x) + s_3, \Phi(v'{:}T) + s_3' - K_4^{\text{app}}) \\
& = & K_1^{\text{app}} \cdot (v + \Phi(V{:}\Gamma_1) + s_1, v')
\end{array}
$$

for $v, v' \in \mathbb{Q}_0^+$ with

$$
\begin{array}{rcl}
v & \leq & \Phi(V{:}\Gamma_2) + s_2 + \Phi(V'[x \mapsto v] : \Gamma', x{:}T_x) + s_3 \\
& & -(\Phi((\lambda x.t'', V'){:}(x{:}T_x \xrightarrow{s_3/s_3'} T)) + s_1' - K_2^{\text{app}} + \Phi(v{:}T_x) + s_2' - K_3^{\text{app}}) \\
& = & \Phi(V{:}\Gamma_2) + s_2 - (s_1' - K_2^{\text{app}}) \\
& = & \Phi(V{:}\Gamma_2) + s_2 + \Phi(v{:}T_x) + s_3 \\
& & -(s_1' - K_2^{\text{app}} + \Phi(v{:}T_x) + s_2' - K_3^{\text{app}}) \\
& = & \Phi(V{:}\Gamma_2) + \Phi(v{:}T_x) - \Phi(v{:}T_x) + s_2 - (s_1' - K_2^{\text{app}}) + s_3 - (s_2' - K_3^{\text{app}}) \\
& \overset{(3.31,3.32)}{=} & \Phi(V{:}\Gamma_2)
\end{array}
$$

and thus

$$
\begin{array}{rcl}
u & \leq & \max(0, \Phi(V{:}\Gamma_1) + \Phi(V{:}\Gamma_2) + s_1 + K_1^{\text{app}}) \\
& \overset{(3.30)}{\leq} & \Phi(V{:}\Gamma) + q
\end{array}
$$

Finally, it follows by applying Proposition 1 to (3.35), (3.37), and (3.29) that $u \geq p$.

For the second part of the statement we apply the properties of watermarks to (3.29) and derive the following. For brevity we let $v'' = (\lambda x.t'', V')$, $T' = (x{:}T_x \xrightarrow{s_3/s_3'} T)$.

$$
\begin{array}{rcl}
p - p' & = & r - r' + s - s' + t - t' + K_1^{\text{app}} + K_2^{\text{app}} + K_3^{\text{app}} + K_4^{\text{app}} \\
& \overset{(3.40,\ 3.38,\ 3.36)}{\leq} & \Phi(V{:}\Gamma_1) + s_1 - (\Phi(v''{:}T') + s_1') + \Phi(V{:}\Gamma_2) + s_2 \\
& & -(\Phi(v{:}T_x) + s_2') + \Phi(V'[x \mapsto v] : \Gamma', x{:}T_x) + s_3 - (\Phi(v'{:}T) + s_3') \\
& & +K_1^{\text{app}} + K_2^{\text{app}} + K_3^{\text{app}} + K_4^{\text{app}} \\
& = & (\Phi(V{:}\Gamma_1) + \Phi(V{:}\Gamma_2) + s_1) - \Phi(v''{:}T') + \Phi(V'{:}\Gamma') + \Phi([x \mapsto v] : x{:}T_x) \\
& & -\Phi(v{:}T_x) + (s_2 + K_2^{\text{app}} - s_1') + (s_3 + K_3^{\text{app}} - s_2') \\
& & -(\Phi(v'{:}T) + s_3') + K_1^{\text{app}} + K_4^{\text{app}} \\
& = & (\Phi(V{:}\Gamma_1) + \Phi(V{:}\Gamma_2) + s_1) \\
& & +(s_2 + K_2^{\text{app}} - s_1') + (s_3 + K_3^{\text{app}} - s_2') \\
& & -(\Phi(v'{:}T) + s_3') + K_1^{\text{app}} + K_4^{\text{app}} \\
& \overset{(3.31,\ 3.32)}{=} & \Phi(V{:}\Gamma_1) + \Phi(V{:}\Gamma_2) + s_1 - (\Phi(v'{:}T) + s_3') + K_1^{\text{app}} + K_4^{\text{app}} \\
& = & \Phi(V{:}\Gamma_1) + \Phi(V{:}\Gamma_2) + s_1 + K_1^{\text{app}} - (\Phi(v'{:}T) + s_3' - K_4^{\text{app}}) \\
& \overset{(3.30,\ 3.33)}{\leq} & \Phi(V{:}\Gamma) + q - (\Phi(v'{:}T) + q')
\end{array}
$$

(LT:Match): Assume that the type derivation of e ends with an application of the rule LT:Match. Then t is a pattern match of the form $\texttt{match}(e; t_1; x_1, x_2.t_2)$, the evaluation of which ends with an application of either E:MatchC or E:MatchN. The latter case is similar to the case for (LT:Cond), so we assume the derivation of the evaluation judgement ends with an application of E:MatchC.

Then $V \vdash e \Downarrow \texttt{cons}(v_1, v_2) \mid (r_1, r_1')$ and $V' \vdash t_2 \Downarrow v \mid (r_2, r_2')$ for $V' = V[x_1 \mapsto v_1, x_2 \mapsto v_2]$ and some r_1, r_1', r_2, r_2' with

$$(p, p') = K^{\text{mat}} \cdot (r_1, r_1') \cdot K_1^{\text{matC}} \cdot (r_2, r_2') \cdot K_2^{\text{matC}} \tag{3.41}$$

Since the derivation of the typing judgement ends with LT:Match, we have $\Gamma \mid\mid \Gamma_1; \Gamma_2,\ \Gamma_1; \Delta \left|\frac{s_1}{s_1'}\right._{\mathbb{Q}} e : \exists \bar{z}{:}\bar{T}_z.\{L^t(T') \mid \psi\},\ \Gamma_2, x_1{:}T', x_2{:}\{L^t(T') \mid \psi_{\texttt{cons}}\}; \Delta, [x'/\nu]g_{\texttt{cons}}, [x'/\nu]\psi \left|\frac{s_2}{s_2'}\right._{\mathbb{Q}} t_2 : T$, and

$$q = s_1 + K^{\text{mat}} \tag{3.42}$$

$$s_1' = s_2 - t + K_1^{\text{matC}} \tag{3.43}$$

$$q' = s_2' - K_2^{\text{matC}} \tag{3.44}$$

Since $\Delta \vdash_{\mathbb{Q}} V : \Gamma_1$ holds by Lemma 2, we can apply the I.H. to $V \vdash e \Downarrow \texttt{cons}(v_1, v_2) \mid (r_1, r_1')$ and get

$$r_1 \leq \Phi(V : \Gamma) + s_1 \tag{3.45}$$

$$r_1 - r_1' \leq \Phi(V : \Gamma) + s_1 - (\Phi(\texttt{cons}(v_1, v_2){:}\bar{z}{:}\bar{T}_z.\{L^t(T') \mid \psi\}) + s_1') \tag{3.46}$$

By Theorem 1 we have that $\Delta \vdash_{\mathbb{Q}} \texttt{cons}(v_1, v_2){:}\bar{z}{:}\bar{T}_z.\{L^t(T') \mid \psi\}$, and by inversion on the value checking rules, we get that $\Delta \vdash_{\mathbb{Q}} v_1{:}T'$ and $\Delta, x_1{:}|T'| \vdash_{\mathbb{Q}} v_2{:}\{L^t(T') \mid \psi_{\texttt{cons}}\}$. We can thus derive that

$$\Delta, x_1{:}|T'|, x_2{:}\{L(|T'|) \mid \psi_{\texttt{cons}}\} \vdash_{\mathbb{Q}} V[x_1 \mapsto v_1, x_2 \mapsto v_2] : \Gamma_2, x_1{:}T', x_2{:}\{L^t(T') \mid \psi_{\texttt{cons}}\}$$

and can therefore apply the I.H. again to the remaining premises and get that

$$r_2 \leq \Phi(V' : \Gamma_2') + s_2 \tag{3.47}$$

$$r_2 - r_2' \leq \Phi(V' : \Gamma_2') + s_2 - (\Phi(v{:}T) + s_2') \tag{3.48}$$

where $\Gamma_2' = \Gamma_2, x_1{:}T', x_2{:}\{L^t(T') \mid \psi_{\texttt{cons}}\}$. Note that $\Phi(V : \Gamma) - t \geq 0$ and let

$$(u, u') = K^{\text{mat}} \cdot (\Phi(V : \Gamma_1) + s_1, \Phi(\texttt{cons}(v_1, v_2){:}\bar{z}{:}\bar{T}_z.\{L^t(T') \mid \psi\}) + s_1') \cdot K_1^{\text{matC}} \cdot (\Phi(V : \Gamma_2') + s_2, \Phi(v{:}T) + s_2') \cdot K_2^{\text{matC}} \tag{3.49}$$

Then it follows that

$$
\begin{array}{rcl}
(u, u') & \overset{(3.43,\ 3.44)}{=} & K^{\mathrm{con}} \cdot (\Phi(\mathtt{cons}(v_1, v_2){:}\bar{z}{:}\bar{T}_z.\{L^t(T') \mid \psi\}) + s_1' - K_1^{\mathrm{matC}}) \cdot \\
& & (\Phi(V : \Gamma_2') + s_2, \Phi(v : T) + s_2' - K_2^{\mathrm{matC}}) \\
& = & K^{\mathrm{con}} \cdot (v + \Phi(V{:}\Gamma_1) + s_1, v')
\end{array}
$$

for $v, v' \in \mathbb{Q}_0^+$ with

$$
\begin{array}{rcl}
v & \leq & \Phi(V{:}\Gamma_2') + s_2 - (\Phi(\mathtt{cons}(v_1, v_2){:}\bar{z}{:}\bar{T}_z.\{L^t(T') \mid \psi\}) + s_1' - K_1^{\mathrm{matC}}) \\
& = & \Phi(V{:}\Gamma_2) + \Phi(v_1{:}T') + \Phi(v_2{:}\{L^t(T') \mid \psi_{\mathtt{cons}}\}) + s_2 \\
& & -(t + \Phi(v_1{:}T') + \Phi(v_2{:}\{L^t(T') \mid \psi_{\mathtt{cons}}\}) + s_1' - K_1^{\mathrm{matC}}) \\
& \overset{(3.43)}{=} & \Phi(V{:}\Gamma_2)
\end{array}
$$

and thus

$$
\begin{array}{rcl}
u & \leq & \max(0, \Phi(V{:}\Gamma_1) + \Phi(V{:}\Gamma_2) + s_1 + K^{\mathrm{mat}}) \\
& \overset{(3.42)}{\leq} & \Phi(V{:}\Gamma) + q
\end{array}
$$

Finally, it follows by applying Proposition 1 to (3.25), (3.27), and (3.21) that $u \geq p$.

For the second part of the statement we apply the properties of watermarks to (3.21) and derive the following.

$$
\begin{array}{rcl}
p - p' & = & r_1 - r_1' + r_2 - r_2' + K^{\mathrm{mat}} + K_1^{\mathrm{matC}} + K_2^{\mathrm{matC}} \\
& \overset{(3.48,\ 3.46)}{\leq} & \Phi(V : \Gamma_1) + s_1 - (\Phi(\mathtt{cons}(v_1, v_2){:}\bar{z}{:}\bar{T}_z.\{L^t(T') \mid \psi\}) + s_1') \\
& & +\Phi(V : \Gamma_2') + s_2 - (\Phi(v : T) + s_2') + K^{\mathrm{mat}} + K_1^{\mathrm{matC}} + K_2^{\mathrm{matC}} \\
& = & (\Phi(V{:}\Gamma_1) + \Phi(V{:}\Gamma_2') + s_1) - \Phi(v_1{:}T') - \Phi(v_2{:}L^t(T')) \\
& & +(s_2 + K_1^{\mathrm{matC}} - t - s_1') - (\Phi(v{:}T) + s_2') + K^{\mathrm{mat}} + K_2^{\mathrm{matC}} \\
& \overset{(3.43)}{=} & (\Phi(V{:}\Gamma_1) + \Phi(V{:}\Gamma_2') + s_1) - \Phi(v_1{:}T') - \Phi(v_2{:}L^t(T')) \\
& & +(\Phi(v{:}T) + s_2') + K^{\mathrm{mat}} + K_2^{\mathrm{matC}} \\
& = & (\Phi(V{:}\Gamma_1) + \Phi(V{:}\Gamma_2) + s_1) + \Phi(v_1{:}T') + \Phi(v_2{:}L^t(T')) \\
& & -\Phi(v_1{:}T') - \Phi(v_2{:}L^t(T')) + (\Phi(v{:}T) + s_2') + K^{\mathrm{mat}} + K_2^{\mathrm{matC}} \\
& = & \Phi(V{:}\Gamma) + s_1 + K^{\mathrm{mat}} - (\Phi(v{:}T) + s_2' - K_2^{\mathrm{matC}}) \\
& \overset{(3.42,\ 3.44)}{\leq} & \Phi(V{:}\Gamma) + q - (\Phi(v{:}T) + q')
\end{array}
$$

□

3.2.3 Soundness and Completeness of Round-Trip Type Checking

In this section, we show that all three type systems are connected by proving that the bidirectional and bottom-up systems are sound and complete with respect to each other and then showing that the bidirectional

and round-trip systems are sound and complete with respect to each other. This demonstrates that the correctness properties we proved about the bottom-up system in Theorems 1 and 2 also apply to the other two type systems.

Lemma 4 (Bidirectional to Bottom-up)**.** *If* $\vdash_{\mathbb{Q}} \Delta$*,* $\Delta \vdash_{\mathbb{Q}} T$*, and* $\Gamma; \Delta \left|\frac{q}{q'}_{\mathbb{Q}}\right. t \downarrow_b T$ *then* $\Gamma; \Delta \left|\frac{q}{q'}_{\mathbb{Q}}\right. t : T'$ *and* $\Delta \vdash T' <: T$*; additionally, if* t *is an E-term and* $\Gamma; \Delta \left|\frac{q}{q'}_{\mathbb{Q}}\right. t \uparrow_b T$ *then* $\Gamma; \Delta \vdash_{\mathbb{Q}} t : T$*.*

Proof. We go by induction on the structure of the derivation. We first assume that t is some E-term e and build an exact derivation $\Gamma; \Delta \left|\frac{q}{q'}_{\mathbb{Q}}\right. e : T$. The cases for BD:Var-Base, BD:Var, BD:True, BD:False, BD:Int, BD:Nil, and BD:WeakInf are trivial.

For the rule BD:Cons, we have by our induction hypothesis that $\Gamma_1; \Delta \left|\frac{q-K_1^{\text{cons}}}{p}_{\mathbb{Q}}\right. e_1 : T$ and

$$\Gamma_2; \Delta, x_1{:}|T| \left|\frac{p-K_2^{\text{cons}}}{q'+r+K_3^{\text{cons}}}_{\mathbb{Q}}\right. e_2 : T'$$

By our third premise and LT:Sub we get that $\Gamma_2; \Delta, x_1{:}|T| \left|\frac{p-K_2^{\text{cons}}}{q'+r+K_3^{\text{cons}}}_{\mathbb{Q}}\right. e_2 : \{L^r(T) \mid \psi_{\texttt{cons}}\}$ and can apply LT:Cons to get the desired result.

The rule BD:AppFO corresponds directly with the derived rule LT:AppDeriv, so we can similarly trivially substitute this case. Lastly, we get by applying the induction hypothesis to the third premise of BD:AppHO, $\Gamma; \Delta \left|\frac{p-K_2^{\text{app}}}{r}_{\mathbb{Q}}\right. f \downarrow_b T_x$, and get T'_x such that $\Gamma; \Delta \left|\frac{p-K_2^{\text{app}}}{r}_{\mathbb{Q}}\right. f : T'_x$ and $\Delta \vdash T'_x <: T_x$. These two results give us the desired premises of LT:AppDeriv. We note that we get $\Delta \vdash_{\mathbb{Q}} T_x$ from $\vdash_{\mathbb{Q}} \Delta$ since $f \in \text{dom}(\Delta)$.

Otherwise, if t is not necessarily an E-term, we case on the type checking rules. For BD:IE, we get by applying the induction hypothesis on the first premise that $\Gamma; \Delta \left|\frac{q}{q'}_{\mathbb{Q}}\right. t : T'$ as desired and get our other desired conclusion from the second premise. For the BD:Fun rule, given a derivation of $\Gamma; \Delta \left|\frac{q+K^{\text{fun}}}{q}_{\mathbb{Q}}\right. \lambda y.t \downarrow_b (x{:}T_x \xrightarrow{p/p'} T)$, we aim to construct a derivation of $\Gamma; \Delta \left|\frac{q}{q'}_{\mathbb{Q}}\right. \lambda y.t : (x{:}T_x \xrightarrow{p/p'} T)$. By assumption, we have that $\Delta \vdash_{\mathbb{Q}} (x{:}T_x \xrightarrow{p/p'} T)$. Furthermore, we apply the I.H. to the first premise of BD:Fun (by way of $\Delta, y{:}|T_x| \vdash_{\mathbb{Q}} [y/x]T$ from our assumption that $\Delta \vdash_{\mathbb{Q}} (x{:}T_x \xrightarrow{p/p'} T)$) and get that $\Gamma, y{:}T_x; \Delta, y{:}|T_x| \left|\frac{p}{p'}_{\mathbb{Q}}\right. t : T'$, where $\Delta, y{:}|T_x| \vdash T' <: [y/x]T$. By using the LT:Sub rule we get that $\Gamma, y{:}T_x; \Delta, y{:}|T_x| \left|\frac{p}{p'}_{\mathbb{Q}}\right. t : [y/x]T$. Finally, by LT:Fun, we get that $\Gamma; \Delta \left|\frac{q+K^{\text{fun}}}{q}_{\mathbb{Q}}\right. \lambda y.t : (x{:}T_x \xrightarrow{p/p'} T)$ as desired. All other type checking cases are analogous. We note that each case conserves the potential annotations between the two judgements; therefore, the resource analysis remains unchanged. □

Lemma 5 (Bottom-up to Bidirectional)**.** *If* $\Gamma; \Delta \left|\frac{q}{q'}_{\mathbb{Q}}\right. t : T$ *then* $\Gamma; \Delta \left|\frac{q}{q'}_{\mathbb{Q}}\right. t \downarrow_b T$*; additionally, if* t *is an E-term, then* $\Gamma \left|\frac{q}{q'}_{\mathbb{Q}}\right. t \uparrow_b T$*.*

Proof. We go by induction on the structure of the derivation. We again first assume that t is some E-term e; we will build a derivation of $\Gamma; \Delta \left|\frac{q}{q'}_{\mathbb{Q}}\right. e \uparrow_b T$. The cases for LT:Var-Base, LT:Var, LT:True, LT:False,

LT:Int, LT:Nil, and LT:Weaken are trivial. In the case for LT:Cons, we get by the induction hypothesis that $\Gamma_2; \Delta, x{:}|T| \left|\frac{p-K_2^{\text{cons}}}{q'+r+K_3^{\text{cons}}}\right._{\mathbb{Q}} e_2 \uparrow_b \{L^r(T) \mid \psi_{\texttt{cons}}\}$. Since $<:$ is reflexive, we get $\Delta, x{:}|T| \vdash \{L^r(T) \mid \top\} <: \{L^r(T) \mid \top\}$ as our desired third premise. For LT:AppDeriv, we have to case on whether the argument t is an E-term or a function term. In the former case, the new derivation is built in a straightforward way. Otherwise, we have from the premises that $\Gamma_2; \Delta \left|\frac{p-K_2^{\text{app}}}{r}\right._{\mathbb{Q}} t : T'_x$ and $\Delta \vdash T'_x <: T_x$. By LT:Sub, we get that $\Gamma_2; \Delta \left|\frac{p-K_2^{\text{app}}}{r}\right._{\mathbb{Q}} t : T_x$, thereby giving us the necessary premise for BD:AppHO by the induction hypothesis.

For all other rules, we use the corresponding bidirectional checking rule in the target derivation, since the premises of the bidirectional rules are a subset of those for the bottom-up rules. We note that each case conserves the potential annotations between the two judgements; therefore, the resource analysis remains unchanged. □

Lemma 6 (Type Strengthening)**.** *If* $\Gamma; \Delta \left|\frac{q}{q'}\right._{\mathbb{Q}} t \downarrow T \uparrow T'$ *then* $\Delta \vdash T' <: T$.

Proof. We go by induction on the structure of the derivation. The cases for RT:Var-Base, RT:Var, RT:True, RT:False, RT:Int, RT:Nil, RT:Cons, and RT:AppFO are trivial, since they include the desired subtyping check as a premise. For RT:AppHO, we have by applying the induction hypothesis to the first premise that

$$\Delta \vdash (T'_x \xrightarrow{r-K_3^{\text{app}}/q'+K_4^{\text{app}}} T') <: (\mathsf{bot} \xrightarrow{r-K_3^{\text{app}}/q'+K_4^{\text{app}}} T)$$

and therefore by the definition of subtyping we get that $\Delta \vdash T' <: T$. □

Lemma 7 (Round-trip to Bidirectional)**.** *If* $\Gamma; \Delta \left|\frac{q}{q'}\right._{\mathbb{Q}} t \downarrow T$ *then* $\Gamma; \Delta \left|\frac{q}{q'}\right._{\mathbb{Q}} t \downarrow_b T$*; additionally, if* t *is an E-term and* $\Gamma; \Delta \left|\frac{q}{q'}\right._{\mathbb{Q}} t \downarrow T \uparrow T'$*, then* $\Gamma; \Delta \left|\frac{q}{q'}\right._{\mathbb{Q}} t \uparrow_b T'$.

Proof. As usual, we first assume an E-term e and a derivation of $\Gamma; \Delta \left|\frac{q}{q'}\right._{\mathbb{Q}} e \downarrow T \uparrow T'$; we seek to build a derivation of $\Gamma; \Delta \left|\frac{q}{q'}\right._{\mathbb{Q}} e \uparrow_b T'$. The cases for RT:Var-Base, RT:Var, RT:True, RT:False, RT:Int, and RT:Nil are trivial.

For RT:Cons, we apply the induction hypothesis and get that $\Gamma_1; \Delta \left|\frac{q-K_1^{\text{cons}}}{p}\right._{\mathbb{Q}} e_1 \uparrow_b T'$ and

$$\Gamma_2; \Delta \left|\frac{p-K_2^{\text{cons}}}{q'+r+K_3^{\text{cons}}}\right._{\mathbb{Q}} e_2 \uparrow_b T''$$

By applying Lemma 6 to the third premise we get that $\Delta, x_1{:}|T'| \vdash T'' <: \{L^r(T') \mid \psi_{\texttt{cons}}\}$ and get the third desired premise.

Rule RT:AppHO has the same structure as BD:AppHO, so we can simply replace instances of one with the other. The case for RT:AppFO follows in the same way as for RT:Cons. Lastly, the case for RT:WeakInf follows directly from applying the induction hypothesis and BD:WeakInf.

For most of the checking rules, the claim follows from applying the induction hypothesis and the corresponding bidirectional checking rule. For RT:IE, we get the first premise by applying the induction hypothesis, and we get the second premise from applying Lemma 6 to the first premise. We note that each case conserves the potential annotations between the two judgements; therefore, the resource analysis remains unchanged. □

Lemma 8 (Bidirectional to Round-Trip)**.** *If* $\Gamma; \Delta \left|\frac{q}{q'}_{\mathbb{Q}}\right. t \downarrow_b T$ *then* $\Gamma; \Delta \left|\frac{q}{q'}_{\mathbb{Q}}\right. t \downarrow T$*; additionally, if* t *is an E-term and* $\Gamma; \Delta \left|\frac{q}{q'}_{\mathbb{Q}}\right. t \uparrow_b T$ *and there exists a type* U *such that* $\Delta \vdash_{\mathbb{Q}} U$ *and* $\Delta \vdash T <: U$ *then* $\Gamma; \Delta \left|\frac{q}{q'}_{\mathbb{Q}}\right. t \downarrow U \uparrow T$.

Proof. We go by induction on the structure of the derivation. We first assume an E-term e and a type U such that $\Gamma; \Delta \left|\frac{q}{q'}_{\mathbb{Q}}\right. e \uparrow_b T$, $\Delta \vdash_{\mathbb{Q}} |U|$, and $\Delta \vdash T <: U$; we seek to build a derivation of $\Gamma; \Delta \left|\frac{q}{q'}_{\mathbb{Q}}\right. e \downarrow U \uparrow T$. The cases for BD:Var-Base, BD:Var, BD:True, BD:False, BD:Int, and BD:Nil are trivial.

For BD:Cons, we get by definition that $\Delta \vdash T <: \mathsf{top}$ and can therefore get the desired second premise. Furthermore, we get from the fourth premise of the bidirectional rule that $\Delta, x_1{:}|T| \vdash T' <: \{L^r(T) \mid \psi_{\mathtt{cons}}\}$, so we can pick the latter type as our goal type and apply the induction hypothesis. Lastly, we get from our assumption on U from the lemma statement that $\Delta \vdash \exists x_1{:}|T|.x_2{:}\{L^r(|T|) \mid \psi_{\mathtt{cons}}\}.\{L^r(T) \mid g_{\mathtt{cons}}\} <: U$, so we get the fourth premise as desired.

For BD:AppFO, we apply the induction hypothesis to the second premise, using the function type $x{:}\{B \mid \bot\} \xrightarrow{r-K_3^{\mathrm{app}}/q'+K_4^{\mathrm{app}}} U$ as our goal. In order to use this, we require that $\Delta \vdash_{\mathbb{Q}} x{:}\{B \mid \bot\} \to U$, which follows since $\Delta \vdash_{\mathbb{Q}} U$. In addition, we require the following to hold:

$$\Delta \vdash x{:}\{B \mid \psi\} \xrightarrow{r-K_3^{\mathrm{app}}/q'+K_4^{\mathrm{app}}} T <: x{:}\{B \mid \bot\} \xrightarrow{r-K_3^{\mathrm{app}}/q'+K_4^{\mathrm{app}}} U$$

By the definition of function subtyping, this simplifies to $\Delta \vdash \{B \mid \bot\} <: \{B \mid \psi\}$ (which follows trivially), and $\Delta, x{:}\{B \mid \bot\} \vdash T <: U$. We have by assumption that $\Delta \vdash \exists x{:}|T_x|.T <: U$, and therefore that $\Delta, x{:}|T_x| \vdash T <: U$. Therefore, we seek to show that $[\![\Delta, x{:}\{B \mid \bot\}]\!]_{T<:U} \Rightarrow [\![\Delta, x{:}|T_x|]\!]_{T<:U}$. If x does not appear in T, then both formulae are equivalent to $[\![\Delta]\!]_{T<:U}$ and the implication holds. Otherwise, one of the conjuncts in the left hand side is $\bot$, thus allowing the implication to hold. To get the third desired premise, we pick $\{B \mid \psi\}$ as our goal. Since it is inferred as the domain of the type of e_1, we know that it is well-formed in Δ; moreover, we get that $\Delta \vdash T_x <: \{B \mid \psi\}$ from the fourth premise of the bidirectional rule. Therefore, we apply the induction hypothesis to the third premise of the bidirectional rule to get the desired result. Lastly, we get the fourth premise from our assumption about U for this lemma.

We now consider the BD:AppHO rule. We get the second premise of the round-trip rule in a similar manner to the case for BD:AppFO; we aim to show that $\Delta \vdash T_x \xrightarrow{r-K_3^{\mathrm{app}}/q'+K_4^{\mathrm{app}}} T <: \mathsf{bot} \xrightarrow{r-K_3^{\mathrm{app}}/q'+K_4^{\mathrm{app}}}$

U. $\Delta \vdash \mathsf{bot} <: T_x$ holds by definition, and we get $\Delta \vdash T <: U$ from our assumption on U from the statement of the lemma. We get the third premise directly from the induction hypothesis.

For the rule BD:IE, we have that $\Delta \vdash T' <: T$ from the premise, so we apply the induction hypothesis with $U = T$ and get the desired premise.

The rest of the rules are trivial, since they have the same shape in both systems. The only complication is the second premise of BD:If and BD:Match; for these, $\{\mathsf{Bool} \mid \top\}$ and top, respectively, are supertypes of any possible inferred type in these premises, so we get our desired premise. We note that each case conserves the potential annotations between the two judgements; therefore, the resource analysis remains unchanged. □

Theorem 3 (Soundness of Round-Trip Type Checking)**.** *If* $\vdash_{\mathbb{Q}} \Delta$, $\Delta \vdash_{\mathbb{Q}} T$, *and* $\Gamma; \Delta \left|^{q}_{q'}{}_{\mathbb{Q}}\right. t \downarrow T$, *then* $\Gamma; \Delta \left|^{q}_{q'}{}_{\mathbb{Q}}\right. t : T'$ *and* $\Gamma \vdash T' <: T$.

Proof. Straightforward by combining Lemma 7 and Lemma 4. □

Theorem 4 (Completeness of Round-Trip Type Checking)**.** *If* $\Gamma; \Delta \left|^{q}_{q'}{}_{\mathbb{Q}}\right. t : T$, *then* $\Gamma; \Delta \left|^{q}_{q'}{}_{\mathbb{Q}}\right. t \downarrow T$.

Proof. Straightforward by combining Lemma 5 and Lemma 8. □

Chapter 4

Analysis

To demonstrate the effectiveness of our contributions, we will walk through two example functions: `append` and `compress`. We include the former since it allows for a fairly straightforward yet thorough assessment of the functionality of the round-trip system. In contrast, the latter serves as an example of a program where our system generates a better program than Synquid, thereby directly showing our improvements. For each example, we will show the Synquid specification, the program generated by the Synquid tool, the annotated specification in our system, a detailed walkthrough of how our solution is generated, and our solution (if it differs from the Synquid solution). In these examples, we seek to bound the number of `cons` operations that occur; however, the resource metric can be instantiated to analyze the usage of a range of resources, including heap space and evaluation steps. Additionally, for readability, we abbreviate types of the form $\{B \mid \top\}$ where B is a base type as simply B. During the execution of the actual synthesis algorithm, as detailed in [20], many more terms are attempted and rejected than discussed here. In these walkthroughs, we focus on exploring the correct implementation and how we verify its correctness.

4.1 Append

Specification

```
append ::  xs:  List a -> ys:  List a -> {List a | len _v == len xs + len ys}
```

This specification describes a function that takes in an arbitrary list `xs` and an arbitrary list `ys` and returns a list with length equal to the sum of the lengths of the input lists. We note that this does not directly specify the contents or order of the output list but instead depends on the order of term enumeration in the Synquid implementation to produce the correct program.

Synquid Solution

From the above specification, Synquid generates the expected implementation of `append`.

```
append = \xs . \ys .
    match xs with
      Nil -> ys
      Cons x3 x4 -> Cons x3 (append x4 ys)
```

Annotated Specification

$$\mathsf{append} : ys{:}L^0(\mathsf{Int}) \xrightarrow{0/0} xs{:}L^1(\mathsf{Int}) \xrightarrow{0/0} \{L^0(\mathsf{Int}) \mid \mathsf{len}\ \nu = \mathsf{len}\ xs + \mathsf{len}\ ys\}$$

We note that the order of arguments to the function must be switched, since in our formulation, only the final argument applied to a function may carry potential. In the future, we hope to alleviate this by integrating the stack typing system from [12], which would allow all curried arguments to contribute potential. For now, we resort to switching arguments. Furthermore, since `xs` carries potential, it is the only one that can be iterated over and used in `cons` operations, so our synthesis system will still produce the correct function. In fact, since exactly $\mathsf{len}\ \mathtt{xs}$ elements must be added to `ys`, and since each element can only be added once, our system provides more specific guarantees about the functional correctness of the resulting program.

Synthesis Walkthrough

Suppose all costs are 0 except $K_3^{\mathtt{cons}}$, which is 1, and define the guard formulae of our list type as follows: $g_{\mathtt{nil}} = (\mathsf{len}\ \nu = 0)$, $g_{\mathtt{cons}} = (\mathsf{len}\ \nu = 1 + \mathsf{len}\ x_2)$, and $\psi_{\mathtt{cons}} = \top$. Therefore, we seek to build a derivation for the following:

$$\cdot;\cdot \left|^{0}_{0\,\mathbb{Q}}\right. \mathsf{append} \downarrow ys{:}L^0(\mathsf{Int}) \xrightarrow{0/0} xs{:}L^1(\mathsf{Int}) \xrightarrow{0/0} \{L^0(\mathsf{Int}) \mid \mathsf{len}\ \nu = \mathsf{len}\ xs + \mathsf{len}\ ys\}$$

At the top level of the type, we can assume it is a recursive function. So we can infer that the outermost layer of the program will be

$$\mathsf{fix}\ \mathsf{append}.\lambda ys : L^0(\mathsf{Int}).t$$

for some t to be determined, and our new goal is

$$\Gamma, ys{:}L^0(\mathsf{Int}); ys : L(\mathsf{Int}) \left|^{0}_{0\,\mathbb{Q}}\right. t \downarrow xs{:}L^1(\mathsf{Int}) \xrightarrow{0/0} \{L^0(\mathsf{Int}) \mid \mathsf{len}\ \nu = \mathsf{len}\ xs + \mathsf{len}\ ys\}$$

where

$$\Gamma = \mathsf{append}{:}(ys{:}L^0(\mathsf{Int}) \xrightarrow{0/0} xs{:}L^1(\mathsf{Int}) \xrightarrow{0/0} \{L^0(\mathsf{Int}) \mid \mathsf{len}\ \nu = \mathsf{len}\ xs + \mathsf{len}\ ys\})$$

We can now assume the (RT:Abs) rule applies and let $t = \lambda ys : L^0(\mathsf{Int}).t_1$ for some new t_1, leaving us with

$$\Gamma, ys{:}L^0(\mathsf{Int}), xs{:}L^1(\mathsf{Int}); \Delta \left|^{0}_{0\,\mathbb{Q}}\right. t_1 \downarrow \{L^0(\mathsf{Int}) \mid \mathsf{len}\ \nu = \mathsf{len}\ xs + \mathsf{len}\ ys\}$$

where

$$\Delta = ys : L(\mathsf{Int}), xs{:}L(\mathsf{Int})$$

As discussed above, we note that this rule only applies because the order of arguments is switched, thereby allowing the context to carry no potential. Furthermore, we could not have applied the RT:Fix rule a second time, since it requires an empty context.

We can now match on a list, first trying the one that carries potential. This demonstrates a departure from previous synthesis algorithms, which had no such information to use as a heuristic. Even if the synthesis algorithm is not updated to take potential into account, choosing to match on `ys` will not satisfy the cost specification. Thus, we let $t_1 = \mathtt{match}(xs; t_2; x_1, x_2.t_3)$ for some t_2, t_3, We note that this works since we can split our context into

$$\Gamma, ys : L^0(\mathsf{Int}), xs : L^1(\mathsf{Int})$$

and

$$\Gamma, ys : L^0(\mathsf{Int}), xs : L^0(\mathsf{Int})$$

and then have that

$$\Gamma, ys{:}L^0(\mathsf{Int}), xs{:}L^1(\mathsf{Int}); \Delta \left|^{0}_{0\,\mathbb{Q}}\right. xs \downarrow \mathsf{top} \uparrow L^1(\{\mathsf{Int} \mid \nu = xs\})$$

thereby giving us our second premise for the (RT:Match) rule. We try the `nil` branch first, giving us the following:

$$\Gamma, ys{:}L^0(\mathsf{Int}), xs{:}L^0(\mathsf{Int}); \Delta' \left|^{0}_{0\,\mathbb{Q}}\right. t_2 \downarrow \{L^0(\mathsf{Int}) \mid \mathsf{len}\ \nu = \mathsf{len}\ xs + \mathsf{len}\ ys\}$$

where

$$\Delta' = (\Delta, x' = xs, \mathsf{len}\ x' = 0)$$

Since we have no terms that fulfill the specification, we try to infer a type that will fulfill it. Here, we have that $x' = xs$ and $\mathsf{len}\ x' = 0$. From this, we get that $\mathsf{len}\ xs + \mathsf{len}\ ys = \mathsf{len}\ ys$. Therefore, we can infer the type $\{L^0(\mathsf{Int}) \mid \mathsf{len}\ \nu = \mathsf{len}\ xs + \mathsf{len}\ ys\}$ for ys and thus let $t_2 = ys$.

In the other case, we have the following:

$$\Gamma'; \Delta' \mid^{1}_{0\,\mathbb{Q}} t_3 \downarrow \{L^0(\mathsf{Int}) \mid \mathsf{len}\ \nu = \mathsf{len}\ xs + \mathsf{len}\ ys\}$$

where

$$\Gamma' = \Gamma, ys{:}L^0(\mathsf{Int}), xs{:}L^0(\mathsf{Int}), x_1{:}\mathsf{Int}, x_2{:}L^1(\mathsf{Int})$$

and

$$\Delta' = (\Delta, x_1{:}\mathsf{Int}, x_2{:}L(\mathsf{Int}), x' = xs, \mathsf{len}\ x' = 1 + \mathsf{len}\ x_2)$$

We can again enumerate terms and test for subtyping and sufficient potential until we get $t_3 = \texttt{cons}(x_1, \mathsf{append}\ ys\ x_2))$. We note that by matching on the list that carries potential, we are able to pay for the `cons` operation. Additionally, since we can give t_3 the refinement $\mathsf{len}\ \nu = 1 + (\mathsf{len}\ x_2 + \mathsf{len}\ ys)$, and thereby derive that $\mathsf{len}\ \nu = \mathsf{len}\ xs + \mathsf{len}\ ys$, this term satisfies the top-level refinement. Having generated a full program, we see that we have written the same implementation for `append` as Synquid (with the necessary argument switching), but have now verified that it performs a linear number of `cons` operations.

4.2 Compress

Specification

```
compress ::  xs:  List a -> {Clist a | elems _v == elems xs}
```

This specification (which is discussed in more detail in Chapter 1) describes a function that takes in an arbitrary list `xs` and returns a list that has the same set of elements as the input list and contains no adjacent equal elements. Once again, we note that this does not directly specify the order of the output list but instead relies on the implementation of Synquid to produce the expected program.

Synquid Solution

From the above specification, Synquid generates the following implementation of `compress`. This function matches on the input list and the compressed version of the tail of the input. If the two have the same initial element, then the latter is returned; otherwise, the initial head is added onto the latter and returned. We note that due to the second recursive call, this implementation is exponential, and therefore could not be generated by our system, which produces a linear implementation.

```
compress = \xs.
  match xs with
    Nil -> Nil
    Cons x3 x4 ->
      match compress x4 with
        Nil -> Cons x3 Nil
        Cons x10 x11 ->
          if x3 == x10
            then compress x4
            else Cons x3 (Cons x10 x11)
```

Annotated Specification

$$\mathsf{compress} : xs{:}L^2(\mathsf{Int}) \xrightarrow{0/0} \{CL^0(\mathsf{Int}) \mid \mathsf{elems}\ \nu = \mathsf{elems}\ xs\}$$

Just as in the Synquid version of this specification, we use two different list types. Here, we use $L^q(\mathsf{Int})$ to mean an arbitrary list of integers, and we use $CL^q(\mathsf{Int})$ to mean a list of integers where no adjacent elements are equal (i.e., a compressed list). This difference is enforced by the guard formula $\psi_{\texttt{cons}}$, which we define in the walkthrough below.

Synthesis Walkthrough

Suppose all costs are 0 except K_3^{cons}, which is 1, and let the first two guard formulae be defined as follows, where elems is a refinement-level function that maps a list to the set of its elements:

$$g_{\texttt{nil}} = (\mathsf{elems}\ \nu = \{\}) \text{ and } g_{\texttt{cons}} = (\mathsf{elems}\ \nu = \{x_1\} \cup \mathsf{elems}\ x_2)$$

Furthermore, suppose we have the following auxiliary function:

$$\mathsf{eq} \downarrow x{:}\mathsf{Int} \xrightarrow{0/0} y{:}\mathsf{Int} \xrightarrow{0/0} \{\mathsf{Bool} \mid \nu = (x = y)\}$$

Lastly, suppose we have two different list types, following the technique in Vazou, et al. [23]: $L^q(T)$, where $\psi_{\texttt{cons}} = \top$, and $CL^q(T)$, where $\psi_{\texttt{cons}} = \neg(x_1 \in \mathsf{heads}\ \nu)$, where heads is a refinement-level function that maps `nil` to $\{\}$ and $\texttt{cons}(v_1, v_2)$ to $\{v_1\}$. Both list types share the same $g_{\texttt{nil}}$ and $g_{\texttt{cons}}$. Therefore, we seek to build a derivation for the following:

$$\cdot;\cdot \vdash^{\lfloor 0}_{\lfloor 0\,\mathbb{Q}} \mathsf{compress} \downarrow L^2(\mathsf{Int}) \xrightarrow{0/0} \{CL^0(\mathsf{Int}) \mid \mathsf{elems}\ \nu = \mathsf{elems}\ xs\}$$

At the top level, we know from the type that this is a function. Since the context is empty, we can let the outermost layer of the program be

$$\mathsf{fix}\ \mathsf{compress}.\lambda xs : L^2(\mathsf{Int}).t$$

for some t to be determined, and our new goal is

$$\Gamma, xs : L^2(\mathsf{Int}); xs : L(\mathsf{Int}) \left|\begin{smallmatrix}0\\0\end{smallmatrix}\right|_{\mathbb{Q}} t \downarrow \{CL^0(\mathsf{Int}) \mid \mathsf{elems}\ \nu = \mathsf{elems}\ xs\}$$

where

$$\Gamma = \mathsf{compress} : (xs{:}L^2(\mathsf{Int}) \xrightarrow{0/0} \{CL^0(\mathsf{Int}) \mid \mathsf{elems}\ \nu = \mathsf{elems}\ xs\})$$

Now that we have a scalar goal, we can try to use our argument variable in a pattern match so that we have $t = \mathtt{match}(xs; t_1; x_1, x_2.t_2)$. We note that this works since can split our context as follows

$$\Gamma, xs : L^2(\mathsf{Int}) \mid\mid \Gamma, xs : L^2(\mathsf{Int}); \Gamma, xs : L^0(\mathsf{Int})$$

and then have that

$$\Gamma, xs{:}L^2(\mathsf{Int}); xs{:}L(\mathsf{Int}) \left|\begin{smallmatrix}0\\0\end{smallmatrix}\right|_{\mathbb{Q}} xs \downarrow \mathsf{top} \uparrow \{L^2(\mathsf{Int}) \mid \nu = xs\}$$

thereby giving us our second premise for the (RT:Match) rule. For the third premise, we require the following to hold:

$$\Gamma, xs{:}L^0(\mathsf{Int}); xs{:}L(\mathsf{Int}), x' = xs, \mathsf{elems}\ x' = \{\} \left|\begin{smallmatrix}0\\0\end{smallmatrix}\right|_{\mathbb{Q}} t_1 \downarrow \{CL^0(\mathsf{Int}) \mid \mathsf{elems}\ \nu = \mathsf{elems}\ xs\}$$

When enumerating possible terms for t_1, we note that our path conditions give us that $\mathsf{elems}\ xs = \{\}$. Therefore, we let $t_1 = \mathtt{nil}$, thereby satisfying the refinement and the cost requirements. We now consider the other case, letting

$$\Gamma' = \Gamma, xs{:}L^0(\mathsf{Int}), x_1{:}\mathsf{Int}, x_2{:}L^2(\mathsf{Int})$$

and

$$\Delta = (xs{:}L(\mathsf{Int}), x_1{:}\mathsf{Int}, x_2{:}L(\mathsf{Int}), x' = xs, \mathsf{elems}\ x' = \{x_1\} \cup \mathsf{elems}\ x_2)$$

so that

$$\Gamma'; \Delta \left|\begin{smallmatrix}2\\0\end{smallmatrix}\right|_{\mathbb{Q}} t_2 \downarrow \{CL^0(\mathsf{Int}) \mid \mathsf{elems}\ \nu = \mathsf{elems}\ xs\}$$

At this point, no available terms satisfy the refinement and cost restrictions, so we once again pattern match, this time using $\mathsf{compress}\ x_2$ as the term against which we match. We can split our context as

above for the new match, giving $t_2 = \texttt{match}(\mathsf{compress}\ x_2; t_3; x_1', x_2'.t_4)$. Moreover, we note that this works since the following holds:

$$\Gamma'; \Delta \mid^{2}_{2\,\mathbb{Q}} \mathsf{compress}\ x_2 \downarrow \mathsf{top} \uparrow \exists ys{:}\{L(\mathsf{Int}) \mid \nu = x_2\}.\{CL^0(\mathsf{Int}) \mid \mathsf{elems}\ \nu = \mathsf{elems}\ ys\}$$

We now wish to generate a term that satisfies the following

$$\Gamma''; \Delta, ys{:}\{L(\mathsf{Int}) \mid \nu = x_2\}, \mathsf{elems}\ x'' = \mathsf{elems}\ ys, \mathsf{elems}\ x'' = \{\} \mid^{2}_{0\,\mathbb{Q}} t_3 \downarrow \{CL^0(\mathsf{Int}) \mid \mathsf{elems}\ \nu = \mathsf{elems}\ xs\}$$

where

$$\Gamma'' = \Gamma, xs{:}L^0(\mathsf{Int}), x_1{:}\mathsf{Int}, x_2{:}L^0(\mathsf{Int})$$

thereby representing the splitting of Γ'.

At this point, we have the following information: $x' = xs$, $\mathsf{elems}\ x' = \{x_1\} \cup \mathsf{elems}\ x_2$, $ys = x_2$, $\mathsf{elems}\ x'' = \mathsf{elems}\ ys$, and $\mathsf{elems}\ x'' = \{\}$. From this, we can conclude $\mathsf{elems}\ xs = \{x_1\}$, so we want to produce a compressed list with precisely that element. Luckily, we can let $t_3 = \texttt{cons}(x_1, \texttt{nil})$, thereby satisfying the refinement (and properly paying for the `cons` cost). We note that this satisfies $\psi_{\texttt{cons}}$ for the compressed list type, since $x_1 \notin \mathsf{heads}\ \texttt{nil}$ holds.

We now turn to t_4, letting

$$\Gamma''' = \Gamma'', x_1'{:}\mathsf{Int}, x_2'{:}\{CL^0(\mathsf{Int}) \mid \neg(x_1' \in \mathsf{heads}\ \nu)\}$$

and

$$\Delta' = (\Delta, ys{:}\{L(\mathsf{Int}) \mid \nu = x_2\}, x_1'{:}\mathsf{Int}, x_2'{:}\{CL(\mathsf{Int}) \mid \neg(x_1' \in \mathsf{heads}\ \nu)\}, \mathsf{elems}\ x'' = \mathsf{elems}\ ys, \mathsf{elems}\ x'' = \{x_1'\} \cup \mathsf{elems}\ x_2')$$

so that

$$\Gamma'''; \Delta' \mid^{1}_{0\,\mathbb{Q}} t_4 \downarrow \{CL^0(\mathsf{Int}) \mid \mathsf{elems}\ \nu = \mathsf{elems}\ xs\}$$

Here, we have to come up with a value of the compressed list type that fulfills the refinement specification. We cannot use `nil` as above, since this would not have the correct elements; we instead must somehow use x_1, x_1', and x_2'. However, if x_1 and x_1' are the same, we will not need to use both. Thus, we introduce a branch via liquid abduction [20] to provide us with this information, letting $t_4 = (\mathsf{if}\ \mathsf{eq}\ x_1\ x_1'\ \mathsf{then}\ t_5\ \mathsf{else}\ t_6)$. This gives us the following goals (with some abbreviation for readability):

$$\Gamma'''; \Delta', x_1 = x_1' \mid^{2}_{0\,\mathbb{Q}} t_5 \downarrow \{CL^0(\mathsf{Int}) \mid \mathsf{elems}\ \nu = \mathsf{elems}\ xs\}$$

$$\Gamma'''; \Delta', \neg(x_1 = x_1') \,|^2_{0\,\mathbb{Q}}\, t_5 \downarrow \{CL^0(\mathsf{Int}) \mid \mathsf{elems}\ \nu = \mathsf{elems}\ xs\}$$

As a reminder, Δ' gives us the following information:

$$x' = xs \quad \mathsf{elems}\ x' = \{x_1\} \cup \mathsf{elems}\ x_2 \qquad \neg(x_1' \in \mathsf{heads}\ x_2')$$
$$ys = x_2 \qquad \mathsf{elems}\ x'' = \mathsf{elems}\ ys \qquad \mathsf{elems}\ x'' = \{x_1'\} \cup \mathsf{elems}\ x_2'$$

When considering t_5, we can derive that $\mathsf{elems}\ \nu = \{x_1'\} \cup \mathsf{elems}\ x_2'$ must hold for t_5 for t_5 to be well typed. At this point, Synquid sets $t_5 = \mathsf{compress}\ x_2$. However, we have that $x_2{:}L^0(\mathsf{Int})$, and we therefore cannot pay for another call to compress. As a result, our system would reject this choice for t_5.

Instead, we can let $t_5 = \texttt{cons}(x_1', x_2')$, thereby fulfilling both the refinement and cost requirements. Similarly, when considering t_6, we can derive that $\mathsf{elems}\ \nu = \{x_1\} \cup \{x_1'\} \cup \mathsf{elems}\ x_2'$. Therefore, we let $t_6 = \texttt{cons}(x_1, \texttt{cons}(x_1', x_2'))$, again fulfilling both the refinement and cost requirements.

In summary, by following the type system, we derived the following implementation of compress that only performs a linear number of `cons` operations (as opposed to the version generated by Synquid):

$$\begin{aligned}&\mathsf{fix}\ \mathsf{compress}.\lambda xs.\texttt{match}(xs; \texttt{nil};\\ &\qquad x_1, x_2.\texttt{match}(\mathsf{compress}\ x_2; \texttt{cons}(x_1, \texttt{nil});\\ &\qquad\quad x_1', x_2'.\mathsf{if}\ \mathsf{eq}\ x_1\ x_1'\ \mathsf{then}\ \texttt{cons}(x_1', x_2')\ \mathsf{else}\ \texttt{cons}(x_1, \texttt{cons}(x_1', x_2'))))\end{aligned}$$

4.3 Summary

In this chapter, we have compared Synquid to our system using two examples: `append` and `compress`. In the former, our system produced an equivalent implementation to the one generated by Synquid; moreover, the addition of resource usage restrictions allowed for a more precise correctness specification, in that every element in the list `xs` had to be added to `ys` exactly once. In the latter, our technique produced an implementation of `compress` that is significantly more efficient than the one generated by Synquid.

Chapter 5

Conclusions and Future Work

In this thesis, we have improved the Synquid approach to type-directed program synthesis by augmenting it with automatic amortized resource analysis (or AARA). By itself, Synquid is able to efficiently generate verifiably correct programs from a type-based specification by using a round-trip type-checking mechanism. By outfitting this type system with AARA, which builds a system of resource constraints from the type derivation of a program that can then be solved to establish an upper bound on resource consumption, we have allowed users of Synquid to not only specify the functional correctness of their programs but also prescribe their resource usage as well.

For future work, we would like to continue integrating the two systems. Firstly, we would like to add the theoretical work presented here into the main implementation of Synquid. In addition, when generating a subterm for a program, Synquid enumerates candidates in order of size (i.e., number of nodes in the resulting abstract syntax tree). We would like to explore ways that resource analysis could be used to improve this process, wherein subterms that consume fewer resources could be tried before more expensive candidates. However, this would require a notion of local resource analysis (as opposed to the current approach of whole-program analysis), which is an open research area.

We would also like to integrate the stack typing mechanism from [12] so that functions can take multiple arguments that carry potential, thereby allowing us to avoid annoyances like the order of arguments to the `append` function in Section 4.1. Additionally, in its current state, our work requires the resource usage restriction to be specified in terms of the actual resource annotations on the type specification. For better usability, we would like to develop a way to specify that a program should be generated in a particular complexity class, rather than requiring the user to determine the precise amount of potential required.

Program synthesis shows great potential for enabling a new layer of abstraction to be added to the task

of programming. With techniques like Synquid, software developers will be able to specify what they need their programs to do at a high level, rather than always having to deal with lower-level details. However, as opposed to current compiler technology, which encompasses many abstraction layers between the programmer and the machine, the added benefit of programmer efficiency provided by current approaches to program synthesis is substantially offset by the quality of the code that is generated. By building on the contributions we present in this thesis, we believe that program synthesis can one day be an everyday tool for the average software engineer.

Bibliography

[1] Ralph Benzinger. Automated Higher-Order Complexity Analysis. *Theor. Comput. Sci.*, 318(1-2):79–103, 2004.

[2] Quentin Carbonneaux, Jan Hoffmann, and Zhong Shao. Compositional Certified Resource Bounds. In *36th Conference on Programming Language Design and Implementation (PLDI'15)*, 2015. Artifact submitted and approved.

[3] Ezgi Çiçek, Deepak Garg, and Umut A. Acar. Refinement Types for Incremental Computational Complexity. In *24th European Symposium on Programming (ESOP'15)*, pages 406–431, 2015.

[4] Nils Anders Danielsson. Lightweight Semiformal Time Complexity Analysis for Purely Functional Data Structures. In *35th ACM Symp. on Principles Prog. Langs. (POPL'08)*, pages 133–144, 2008.

[5] Norman Danner, Daniel R. Licata, and Ramyaa Ramyaa. Denotational Cost Semantics for Functional Languages with Inductive Types. In *29th Int. Conf. on Functional Programming (ICFP'15)*, 2012.

[6] Norman Danner, Jennifer Paykin, and James S. Royer. A Static Cost Analysis for a Higher-Order Language. In *7th Workshop on Prog. Languages Meets Prog. Verification (PLPV'13)*, pages 25–34, 2013.

[7] John K. Feser, Swarat Chaudhuri, and Isil Dillig. Synthesizing data structure transformations from input-output examples. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '15, pages 229–239, New York, NY, USA, 2015. ACM.

[8] Cormac Flanagan. Hybrid type checking. In *Conference Record of the 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '06, pages 245–256, New York, NY, USA, 2006. ACM.

[9] Bernd Grobauer. Cost Recurrences for DML Programs. In *6th Int. Conf. on Funct. Prog. (ICFP'01)*, pages 253–264, 2001.

[10] Jan Hoffmann. *Types with Potential: Polynomial Resource Bounds via Automatic Amortized Analysis*. PhD thesis, Ludwig-Maximilians-Universität München, 2011.

[11] Jan Hoffmann, Klaus Aehlig, and Martin Hofmann. Multivariate Amortized Resource Analysis. In *38th Symp. on Principles of Prog. Langs. (POPL'11)*, pages 357–370, 2011.

[12] Jan Hoffmann, Ankush Das, and Shu-Chun Weng. Towards Automatic Resource Bound Analysis for OCaml. In *44th Symposium on Principles of Programming Languages (POPL'17)*, 2017.

[13] Martin Hofmann and Steffen Jost. Static Prediction of Heap Space Usage for First-Order Functional Programs. In *30th ACM Symp. on Principles of Prog. Langs. (POPL'03)*, pages 185–197, 2003.

[14] Steffen Jost, Kevin Hammond, Hans-Wolfgang Loidl, and Martin Hofmann. Static Determination of Quantitative Resource Usage for Higher-Order Programs. In *37th ACM Symp. on Principles of Prog. Langs. (POPL'10)*, pages 223–236, 2010.

[15] Etienne Kneuss, Ivan Kuraj, Viktor Kuncak, and Philippe Suter. Synthesis modulo recursive functions. In *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications*, OOPSLA '13, pages 407–426, New York, NY, USA, 2013. ACM.

[16] Kenneth Knowles and Cormac Flanagan. Compositional reasoning and decidable checking for dependent contract types. In *Proceedings of the 3rd Workshop on Programming Languages Meets Program Verification*, PLPV '09, pages 27–38, New York, NY, USA, 2008. ACM.

[17] Ugo Dal Lago and Barbara Petit. The Geometry of Types. In *40th ACM Symp. on Principles Prog. Langs. (POPL'13)*, pages 167–178, 2013.

[18] Peter-Michael Osera and Steve Zdancewic. Type-and-example-directed program synthesis. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '15, pages 619–630, New York, NY, USA, 2015. ACM.

[19] Benjamin C. Pierce and David N. Turner. Local type inference. *ACM Trans. Program. Lang. Syst.*, 22(1):1–44, January 2000.

[20] Nadia Polikarpova, Ivan Kuraj, and Armando Solar-Lezama. Program synthesis from polymorphic refinement types. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '16, pages 522–538, New York, NY, USA, 2016. ACM.

[21] Patrick M. Rondon, Ming Kawaguci, and Ranjit Jhala. Liquid types. In *Proceedings of the 29th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '08, pages 159–169, New York, NY, USA, 2008. ACM.

[22] Pedro Vasconcelos. *Space Cost Analysis Using Sized Types*. PhD thesis, School of Computer Science, University of St Andrews, 2008.

[23] Niki Vazou, Patrick M. Rondon, and Ranjit Jhala. Abstract refinement types. In *Proceedings of the 22Nd European Conference on Programming Languages and Systems*, ESOP'13, pages 209–228, Berlin, Heidelberg, 2013. Springer-Verlag.