Under consideration for publication in J. Functional Programming

A Relational Logic for Higher-Order Programs

ALEJANDRO AGUIRRE
Imdea Software Institute & Universidad Politécnica de Madrid, Spain

GILLES BARTHE
Imdea Software Institute, Spain & MPI-SP, Germany

MARCO GABOARDI
University at Buffalo, SUNY, USA

DEEPAK GARG
MPI-SWS, Germany

PIERRE-YVES STRUB
École Polytechnique, France

Abstract

Relational program verification is a variant of program verification where one can reason about two programs and as a special case about two executions of a single program on different inputs. Relational program verification can be used for reasoning about a broad range of properties, including equivalence and refinement, and specialized notions such as continuity, information flow security or relative cost. In a higher-order setting, relational program verification can be achieved using relational refinement type systems, a form of refinement types where assertions have a relational interpretation. Relational refinement type systems excel at relating structurally equivalent terms but provide limited support for relating terms with very different structures.

We present a logic, called Relational Higher Order Logic (RHOL), for proving relational properties of a simply typed λ-calculus with inductive types and recursive definitions. RHOL retains the type-directed flavour of relational refinement type systems but achieves greater expressivity through rules which simultaneously reason about the two terms as well as rules which only contemplate one of the two terms. We show that RHOL has strong foundations, by proving an equivalence with higher-order logic (HOL), and leverage this equivalence to derive key meta-theoretical properties: subject reduction, admissibility of a transitivity rule and set-theoretical soundness. Moreover, we define sound embeddings for several existing relational type systems such as relational refinement types and type systems for dependency analysis and relative cost, and we verify examples that were out of reach of prior work.

1 Introduction

Many important aspects of program behavior go beyond the traditional characterization of program properties as sets of traces (Alpern & Schneider, 1985). Hyperproperties (Clarkson & Schneider, 2008) generalize properties and capture a larger class of program behaviors, by focusing on sets of sets of traces. As an intermediate point in this space, relational properties are sets of pairs of traces. Relational properties encompass many properties of interest, including program equivalence and refinement, as well as more specific notions such as non-interference and continuity.

Relational verification is an instance of program verification that targets relational properties. Expectedly, standard verification methods such as type systems, program logics, and program analyses can be lifted to a relational setting. However, it remains a challenge to devise sufficiently powerful methods that can be used to verify a broad range of examples. In effect, most existing relational verification methods are limited in the examples that they can naturally verify, due to the fundamental tension between the syntax-directed nature of program verification, and the need to relate structurally different programs. Moreover, approaches to resolve this tension highly depend on the programming paradigm, on the class of program properties considered, and on the verification method. In the (arguably simplest) case of deductive verification of general properties of imperative programs, one approach to reduce this tension is to use self-composition (Barthe *et al.*, 2004), which reduces relational verification to standard verification. However, reasoning about self-composed programs might be cumbersome. Alternatively, there exist expressive relational program logics that rely on an intricate set of rules to reason about a pair of programs. These logics combine two-sided rules, in which the two programs have the same top-level structure, and one-sided rules, which operate on a single program. Rules for loops are further divided into synchronous, in which both programs perform the same number of iterations, and asynchronous rules, that do not have this restriction but introduce more complexity (Benton, 2004; Barthe *et al.*, 2017).

In contrast, deductive verification of general properties of (pure) higher-order programs is less developed. One potential approach to solve the tension between the syntax-directedness, and the need to relate structurally different programs, is to reduce relational verification of pure higher-order programs to proofs in higher-order logic. There are strong similarities between this approach and self-composition: it reduces relational verification to standard verification, but this approach is very difficult to use in practice. A better alternative is to use relational refinement types such as rF* (Barthe *et al.*, 2014), HOARe^2 (Barthe *et al.*, 2015), DFuzz (Gaboardi *et al.*, 2013) or RelCost (Çiçek *et al.*, 2017). Informally, relational refinement type systems use assertions to capture relationships between inputs and outputs of two higher-order programs. They are appealing for two reasons:

- They capture many important properties of programs in a direct and intuitive manner. For instance, the type $\{x :: \mathbb{N} \mid x_1 \leq x_2\} \to \{y :: \mathbb{N} \mid y_1 \leq y_2\}$ captures the set of pairs of functions that preserve the natural order on natural numbers, i.e. pairs of functions $f_1, f_2 : \mathbb{N} \to \mathbb{N}$ such that for every $x_1, x_2 \in \mathbb{N}$, $x_1 \leq x_2$ implies $f_1(x_1) \leq f_2(x_2)$. (The subscripts 1 and 2 on a variable refer to its values in the two runs.)

- They can potentially benefit from a long and successful line of foundational (Freeman & Pfenning, 1991; Xi & Pfenning, 1999; Dunfield & Pfenning, 2004; Melliès & Zeilberger, 2015) and practical (Vazou *et al.*, 2014; Swamy *et al.*, 2016) research on refinement types.

Unfortunately, existing relational refinement type systems fail to support the verification of several examples. Broadly speaking, the two programs in a relational judgment may be required to have the same type and the same control flow; moreover, this requirement must be satisfied by their subprograms: if the two programs are applications, then the two sub-programs in argument position (resp. in function position) must have the same type and the same control flow; if the two programs are case expressions, they must enter the same branch, and their branches must themselves have the same control flow; if the two programs are recursive definitions, then their bodies must perform the same sequence of recursive calls; etc. This restriction, which can be found in more or less strict forms in the different relational type systems, limits the ability to carry fine-grained reasoning about terms that are structurally different. This raises the question whether the type-directed form of reasoning purported by refinement types can be reconciled with an expressive relational verification of higher-order programs. We provide a positive answer for pure higher-order programs; extending our results to effectful programs is an important goal, but we leave it for future work.

Our starting point is the observation that relational refinement type systems are inherently restricted to reasoning about two structurally similar programs, because relational assertions are embedded into types. In order to provide broad support for one-sided rules (i.e., rules that contemplate only one of the two expressions), it is therefore necessary to consider relational assertions at the top-level, since one-sided rules have a natural formulation in this setting. Considering relational assertions at the top-level can be done in two different ways: either by supporting a rich theory of subtyping for relational refinement types, in such a way that each type admits a normal form where refinements only arise at the top-level, or simply by adapting the definitions and rules of refinement type systems so that only the top-level refinements are considered. Although both approaches are feasible, we believe that the second approach is more streamlined and leads to friendlier verification environments.

Contributions

We present a new logic, called Relational Higher Order Logic (RHOL, § 6), for reasoning about relational properties of higher-order programs written in a variant of Plotkin's PCF (§ 2). The logic manipulates judgments of the form:

$$\Gamma \mid \Psi \vdash t_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi$$

where Γ is a simply typed context, σ_1 and σ_2 are (possibly different) simple types, t_1 and t_2 are terms, Ψ is a set of assertions, and ϕ is an assertion. Our logic retains the type-directed nature of (relational) refinement type systems, and features typing rules for reasoning about structurally similar terms. However, disentangling types from assertions also makes it possible to define type-directed rules operating on a single term (left or right) of the judgment. This confers great expressivity to the logic, without significantly affecting its type-directed nature, and opens the possibility to alternate freely between two-sided and one-sided reasoning, as done in the logics for first-order imperative languages.

The validity of judgments is expressed relative to a set-theoretical semantics—our variant of PCF is restricted to terms which admit a set-theoretical semantics, including strongly normalizing terms. More precisely, a judgment $\Gamma \mid \Psi \vdash t_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi$ is valid if for every valuation ρ (mapping variables in the context Γ to elements in the interpretation of their types), the interpretation of ϕ is true whenever the interpretation of (all the assertions in) Ψ is true. Soundness of the logic can be proved through a standard model-theoretic argument; however, we provide an alternative proof based on a sound and complete embedding into Higher-Order Logic (HOL, § 4). We leverage this equivalence to establish several meta-theoretical properties of the logic, notably subject reduction.

Moreover, we demonstrate that RHOL can be used as a general framework, by defining sound embedding for several relational type systems: relational refinement types (§ 7.2), the Dependency Core Calculus (DCC) for many dependency analyses, including those for information flow security (§ 7.3), and the RelCost (§ 7.4) type system for relative cost. The embedding of RelCost is particularly interesting, since it exercises the ability of our logic to alternate between synchronous and asynchronous reasoning. Afterwards, we verify several examples that go beyond the capabilities of previous systems (§ 8). The system, its metatheory and the examples have been mechanized in Coq. We comment on the Coq implementation in Section 9. In Section 10, we conclude by presenting other relational systems based on RHOL that have been developed since the conference version of this paper (Aguirre *et al.*, 2017), and commenting on how they extend RHOL.

Related Work

While dependent type theory is the prevailing approach to reason about (pure) higher-order programs, several authors have explored another approach, which is crisply summarized by Jacobs (1999): "A logic is always a logic over a type theory". Formalisms following this approach are defined in two stages; the first stage introduces a (dependent) type theory for writing programs, and the second stage introduces a predicate logic to reason about programs. This approach has been pursued independently in a series of works on logic-enriched type theories (Aczel & Gambino, 2000; Aczel & Gambino, 2006; Belo, 2007; Adams & Luo, 2010), and on refinement types (Pfenning, 2008; Zeilberger, 2016). In the latter line of work, programs are written in an intrinsically typed λ-calculus à la Church; then, a system of sorts (a.k.a. refinements) is used to establish properties of programs typable in the first system. Our approach is similar; however, these works are developed in a unary setting, and do not consider the problem of relational verification. A further approach consists of developing a logic in an untyped setting, as is the case of LTC (Dybjer, 1985).

Moreover, there is a large body of work on relational verification; we focus on type-based methods and deductive methods. Relational Hoare Logic (Benton, 2004) and Relational Separation Logic (Yang, 2007) are two program logics, respectively based on Hoare Logic and Separation Logic, for reasoning about relational properties of (first-order) imperative programs. These logics have been used for a broad range of examples and applications, ranging from program equivalence to compiler verification and information flow analysis. Moreover, they have been extended in several directions. For example, Probabilistic Relational Hoare Logic (Barthe *et al.*, 2009) and approximate probabilistic Relational Hoare Logic (Barthe *et al.*, 2012) are generalizations of Relational Hoare logic for reasoning about relational properties of (first-order) probabilistic programs. These logics have been used for a broad range of applications, including probabilistic information flow, side-channel security, proofs of cryptographic strength (reductionist security) and differential privacy. Cartesian Hoare Logic (Sousa & Dillig, 2016) is also a recent generalization of Relational Hoare Logic for reasoning about bounded safety (i.e. k-safety for arbitrary but fixed k) properties of (first-order) imperative programs. This logic has been used for analyzing standard libraries. Experiments have demonstrated that such logics can be very effective in practice. Our formalism can be seen as a proposal to adapt their flexibility to pure higher-order programs.

Product programs (Barthe *et al.*, 2004; Terauchi & Aiken, 2005; Zaks & Pnueli, 2008; Barthe *et al.*, 2011) are a general class of constructions that emulate the behavior of two programs and can be used for reducing relational verification to standard verification. While product programs naturally achieve (relative) completeness, they are often difficult to use since they require global reasoning on the obtained program—however recent works (Blatter *et al.*, 2017) show how this approach can be automated in specific settings. Building product programs for (pure) higher-order languages is an intriguing possibility, and it might be possible to instrument RHOL using ideas from (Barthe *et al.*, 2017) to this effect; however, the product programs constructed in (Barthe *et al.*, 2017) are a consequence, rather than a means, of relational verification.

Several type systems have been designed to support formal reasoning about relational properties for functional programs. Some of the earlier works in this direction have focused on the semantics foundations of parametricity, like the work by Abadi *et al.* (1993) on System R, a relational version of System F. The recent work by Ghani *et al.* (2016a) has further extended this approach to give better foundations to a combination of relational parametricity and impredicative polymorphism. Interestingly, similarly to RHOL, System R also supports relations between expressions at different types, although, since System R does not support refinement types, the only relations that System R can support are the parametric ones on polymorphic terms. In RHOL, we do not support parametric polymorphism à la System F currently but the relations that we support are more general. Adding parametric polymorphism will require foregoing the set-theoretical semantics, but it should still be possible to prove equivalence with a polymorphic variant of higher-order logic.

Several type systems have been proposed to reason about information flow security, a prime example of a relational property. Some examples include SLAM (Heintze & Riecke, 1998), the type system underlying Flow Caml (Pottier & Simonet, 2002) and DCC (Abadi *et al.*, 1999). Most of these type systems consider only one expression but they allow the use of information flow labels to specify relations between two different executions of the expression. As we show in this paper, this approach can also be implemented in RHOL. We show how to translate DCC since it is one of the most general type systems; however, similar translations can also be provided for the other type systems.

Relational Hoare Type Theory (RHTT) (Nanevski *et al.*, 2013; Stewart *et al.*, 2013) is a formalism for relational reasoning about stateful higher-order programs. RHTT was designed to verify security properties like authorization and information flow policies but was used for the verification of heterogeneous pointer data structures as well. RHTT uses a monad to separate stateful computations and relational refinements on the monadic type express relational pre- and post-conditions. RHTT supports reasoning about two different programs but the programs must have the same types at the top-level. RHTT's rules support both two- and one-sided reasoning similar to RHOL, but the focus of RHTT is on verifying properties of the program state. In particular, examples such as those in §8 or embeddings such as those in §7 were not considered in RHTT. RHTT is proved sound over a domain-theoretic model and continuity must be proven explicitly during the verification of recursive functions (rules are provided to prove continuity in many cases). In contrast, RHOL's set-theoretic model is simpler, but admits only those recursive functions that have a unique interpretation in set-theory.

Logical relations (Plotkin, 1973; Statman, 1985; Tait, 1967) provide a fundamental tool for reasoning about programs. They have been used for a broad range of purposes, including proving unary properties (for instance strong normalization or complexity) and relational properties (for instance equivalence or information flow security). Our work can be understood as an attempt to internalize the versatility of relational logical relations in a syntactic framework. There is a large body of work on logics for logical relations, from the early work by Plotkin & Abadi (1993) to more recent work on logics for reasoning about states and concurrency (Dreyer *et al.*, 2011; Dreyer *et al.*, 2010; Jung *et al.*, 2015; Krogh-Jespersen *et al.*, 2017). In particular, the IRIS logic (Jung *et al.*, 2015) can be seen as a powerful reasoning framework for logical relations, as shown by Krogh-Jespersen *et al.* (2017). Even though we also aim to internalize logical relations, the goal of RHOL differs from the goal of IRIS in that we aim for syntax-driven relational verification, which IRIS does not.

We have already mentioned the works on relational refinement type systems for verifying cryptographic constructions (Barthe *et al.*, 2014), for differential privacy (Barthe *et al.*, 2015; Gaboardi *et al.*, 2013) and for relational cost analysis (Çiçek *et al.*, 2017). This line of works is probably the most related to our work. However, RHOL improves over all of them, as also shown by some of the embeddings we give in Section 7. Another work in this direction is that of (Asada *et al.*, 2016), which proposes a technique to reduce relational refinement to standard first-order refinements. Their technique is incomplete but it works well on some concrete examples. As discussed earlier, we believe that some technique of this kind can also be applied to RHOL. However, this is orthogonal to our current goal and we leave this investigation to future work.

In a recent paper, Grimm et al. (2018) propose an alternative manner of proving relational properties of monadic computations in F*. Intrinsic specification of monadic computations in this language needs to be unary, however relational properties can be proven extrinsically by using a reification operator. This operator exposes the computational content of the terms and turns them into pure expressions, about which logical lemmas can be written and proven, exploiting the automation features of F*. The idea of exposing the pure representation of a computation generalizes the ideas behind our embeddings of DCC and RelCost. However, this work offers no specific support for relational reasoning and would correspond in our setting to embedding these frameworks directly into HOL rather that into RHOL.

Several works have explored how to automate the verification of higher order programs by means of techniques inspired by model checking and Horn clause solving. Some of the recent works in this direction (Kobayashi *et al.*, 2017; Kobayashi *et al.*, 2018) have focused on the use of Higher-order modal Fixed point Logics (HFL). In this approach, a program is translated to an HFL formula expressing the correctness of the program with respect to a particular property, and the verification is performed via HFL model checking. This approach is able to express several properties of higher order programs in a uniform way. The verification technique based on HFL is quite different from the one we present here but both techniques serve similar purposes. Properties that have been targeted using HFL include reachability, and trace properties, which are all non-relational. Instead, we focused on relational properties including information flow, relational cost, and program equivalence. Nevertheless, it would be interesting to investigate whether HFL model checking could be used also to support the automated verification of relational properties. Program verification based on Horn clauses has been also used for the verification of relational properties of higher order programs (Unno *et al.*, 2017). The technique by Unno et al. focuses on a principled way to encode relational properties in Horn clauses. In our approach instead, the relational nature of the properties is used in the corresponding logic RHOL and its combined use with UHOL and HOL.

Since the publication of the conference version of this article, a few other papers have built on top of RHOL to verify relational properties of effectful programs. The R^c system (Radicek *et al.*, 2018) establishes the relative cost of a pair of higher-order programs. Meanwhile, Guarded RHOL (Aguirre *et al.*, 2018) extends RHOL with guarded recursion and probabilities, which allows reasoning about probabilistic infinite data structures, such as Markov Chains. A recent paper, (Sato *et al.*, 2019), shows another extension of RHOL with continuous probabilities and Bayesian conditioning. We further discuss these extensions in Section 10. These results witness the versatility of the basic principles behind RHOL.

Comparison with conference version

This is an extended version of a conference paper (Aguirre *et al.*, 2017). The increments with respect to the conference version are:

- A mechanization of our system in the Coq proof assistant (§ 9).
- More detailed explanation of the system (§ 4, 5, and 6) and the examples (§ 8). We also add an informal derivation of an example at the beginning of the paper (§ 3), which helps in motivating the theoretical development that follows.
- Detailed proofs of the main theoretical results of this paper.
- A new example (§ 8.3) showing 1-sensitivity of sorting, which we believe to better showcase the advantages of relational reasoning.
- A discussion on extensions of our system that were presented after the publication of the conference version (§ 10).

2 (A variant of) PCF

We consider a variant of PCF (Plotkin, 1977) with booleans, natural numbers, lists, and recursive definitions. For the latter, we require that all recursive calls are performed on strictly smaller elements—as a consequence, the fixpoint equation derived from the definition has a unique set-theoretical solution. The method to enforce this requirement is orthogonal to our design, and could for instance be based on a syntactic guard predicate, or on sized types.

Types and terms of the language are defined by the following grammar:

$$\begin{aligned}
\tau &::= \mathbb{B} \mid \mathbb{N} \mid \mathsf{list}_\tau \mid \tau\times\tau \mid \tau\to\tau \\
t &::= x \mid \langle t,t\rangle \mid \pi_1\ t \mid \pi_2\ t \mid t\ t \mid \lambda x:\tau.t \mid c \mid S\ t \mid t::t \mid \text{case}\ t\ \mathsf{of}\ 0\mapsto t; S\mapsto t \\
&\quad \mid \text{case}\ t\ \mathsf{of}\ \mathsf{tt}\mapsto t;\mathsf{ff}\mapsto t \mid \text{case}\ t\ \text{of}\ []\mapsto t; \_ :: \_ \mapsto t \mid \text{letrec}\ f\ x = t
\end{aligned}$$

where x ranges over a set V of variables, c ranges over the set $\{\mathsf{tt},\mathsf{ff},0,[]\}$ of constants, and λ-abstractions are *à la* Church. Throughout the paper we will use letters t,u,v to range over terms, and letters x,y,z to range over variables. Occasionally, we will also use f,g for variables with arrow types.

The operational behavior of terms is captured by $\beta\iota\mu$-reduction $\to_{\beta\iota\mu} = \to_\beta \cup \to_\iota \cup \to_\mu$, where β-reduction, ι-reduction and μ-reduction are defined as the contextual closure of:

$$\begin{array}{rl}
(\lambda x.t)\ u & \to_\beta t[u/x] \\
\pi_i\langle t_1,t_2\rangle & \to_\beta t_i \\
\text{case}\ 0\ \mathsf{of}\ 0\mapsto u; S\mapsto v & \to_\iota u \\
\text{case}\ St\ \mathsf{of}\ 0\mapsto u; S\mapsto v & \to_\iota (v\ t) \\
\text{case}\ \mathsf{tt}\ \mathsf{of}\ \mathsf{tt}\mapsto u; \mathsf{ff}\mapsto v & \to_\iota u \\
\text{case}\ \mathsf{ff}\ \mathsf{of}\ \mathsf{tt}\mapsto u; \mathsf{ff}\mapsto v & \to_\iota v \\
\text{case}\ []\ \text{of}\ []\mapsto u; \_ :: \_ \mapsto v & \to_\iota u \\
\text{case}\ h::t\ \text{of}\ []\mapsto u; \_ :: \_ \mapsto v & \to_\iota (v\ h\ t) \\
(\text{letrec}\ f\ x = t)\ (C\ \vec{t}) & \to_\mu t[C\ \vec{t}/x][\text{letrec}\ f\ x = t/f]
\end{array}$$

where $t[u/x]$ denotes the usual (capture-free) notion of substitution on terms (replace x by u in t). As usual, we let $=_{\beta\iota\mu}$ denote the reflexive, symmetric, and transitive closure of $\to_{\beta\iota\mu}$. In particular, we only allow reduction of letrec when the argument has a constructor $C \in \{\mathsf{tt}, \mathsf{ff}, 0, S, [], ::\}$ in head position.

Typing judgments are of the form $\Gamma \vdash t : \tau$, where Γ is a set of typing declarations of the form $x : \sigma$, such that each variable is declared at most once. The typing rules are standard, except the one for recursive functions. In this rule, we require that the domain of the recursive function be an inductive type (naturals or lists here) and that the body of the recursive definition letrec $f\ x = t$ satisfy a predicate $\mathcal{D}ef(f,x,t)$ which ensures that all recursive calls are performed on smaller arguments. The typing rule for recursive definitions is thus:

$$\frac{\Gamma, f : I \to \sigma, x : I \vdash e : \sigma \qquad \mathcal{D}ef(f,x,t) \qquad I \in \{\mathbb{N}, \mathsf{list}_\tau\}}{\Gamma \vdash \text{letrec } f\ x\ = t : I \to \sigma}$$

We give set-theoretical semantics to this system. The choice of this model is motivated by simplicity, but our construction would still work in a different model. For instance, the Coq mechanization of this paper uses the Calculus of Inductive Constructions, and the extension of RHOL presented in (Aguirre *et al.*, 2018) has an interpretation in the topos of trees.

For each type τ, its interpretation $[\![\tau]\!]$ is the set of its values:

$$[\![\mathbb{B}]\!] \triangleq \mathbb{B} \quad [\![\mathbb{N}]\!] \triangleq \mathbb{N} \quad [\![\mathsf{list}_\tau]\!] \triangleq \mathsf{list}_{[\![\tau]\!]} \quad [\![\sigma \to \tau]\!] \triangleq [\![\sigma]\!] \to [\![\tau]\!]$$

where $[\![\sigma]\!] \to [\![\tau]\!]$ is the set of total functions with domain $[\![\sigma]\!]$ and codomain $[\![\tau]\!]$.

A valuation ρ for a context Γ (written $\rho \models \Gamma$) is a partial map such that $\rho(x) \in [\![\tau]\!]$ whenever $(x : \tau) \in \Gamma$. For every valuation ρ let $\rho[v/x]$ denote its unique extension ρ' such that $\rho'(y) = v$ if $x = y$ and $\rho'(y) = \rho(y)$ otherwise. Given a valuation ρ for a context Γ, every term t for which a typing judgment $\Gamma \vdash t : \tau$ can be derived, has an interpretation $(\!|t|\!)_\rho$:

$$(\!|x|\!)_\rho \triangleq \rho(x) \qquad (\!|\langle t,u\rangle|\!)_\rho \triangleq \langle (\!|t|\!)_\rho, (\!|u|\!)_\rho\rangle \qquad (\!|\pi_i\ t|\!)_\rho \triangleq \pi_i((\!|t|\!)_\rho)$$

$$(\!|\lambda x : \tau.t|\!)_\rho \triangleq \lambda v : [\![\tau]\!].(\!|t|\!)_{\rho[(\!|v|\!)_\rho/x]} \qquad (\!|c|\!)_\rho \triangleq c \qquad (\!|S\ t|\!)_\rho \triangleq S\ (\!|t|\!)_\rho$$

$$(\!|t :: u|\!)_\rho \triangleq (\!|t|\!)_\rho :: (\!|u|\!)_\rho$$

$$(\!|\text{case } t \text{ of } [] \mapsto u; \_ :: \_ \mapsto v|\!)_\rho \triangleq \begin{cases} (\!|u|\!)_\rho & \text{if } (\!|t|\!)_\rho = [] \\ (\!|v|\!)_\rho\ M\ N & \text{if } (\!|t|\!)_\rho = M :: N \end{cases}$$

$$(\!|\text{letrec } f\ x = t|\!)_\rho \triangleq F$$

In the case of letrec $f\ x = t$, we require that F be the unique solution of the fixpoint equation extracted from the recursive definition—existence and uniqueness of the solution follows from the validity of the $\mathcal{D}ef(f,x,t)$ predicate.

The interpretation of well-typed terms is sound. Moreover, the interpretation equates convertible terms. (This extends to η-conversion.)

Theorem 1 (*Soundness of set-theoretic semantics*)
The following hold:

- If $\Gamma \vdash t : \tau$ and $\rho \models \Gamma$, then $(\!|t|\!)_\rho \in [\![\tau]\!]$.
- If $\Gamma \vdash t : \tau$ and $\Gamma \vdash u : \tau$ and $t =_{\beta\iota\mu} u$ and $\rho \models \Gamma$, then $(\!|t|\!)_\rho = (\!|u|\!)_\rho$.

3 Introductory example

In this section we show a motivating example for our logic. We use an informal presentation style here. Later, in Section 8, we revisit the example and prove it formally. Consider the following pair of programs corresponding to two implementations of the factorial function:

$$\text{fact}_1 \triangleq \text{letrec } f_1\ n_1 = \text{case } n_1 \text{ of } 0 \mapsto 1; S \mapsto \lambda x_1.(S\ x_1) * (f_1\ x_1)$$
$$\text{fact}_2 \triangleq \text{letrec } f_2\ n_2 = \lambda acc.\text{case } n_2 \text{ of } 0 \mapsto acc; S \mapsto \lambda x_2.f_2\ x_2\ ((S\ x_2) * acc)$$

We want to show a relation between the functions, namely that for any natural n, and any initial value a of the accumulator, $a * (\text{fact}_1 n) = \text{fact}_2\ n\ a$. To avoid having to name our programs, we assume our logic has two distinguished variables $\mathbf{r}_1$, $\mathbf{r}_2$, that represent the pair of programs we reason about. Then, the relation we want to show is

$$\forall n. \forall a. a * (\mathbf{r}_1\ n) = \mathbf{r}_2\ n\ a$$

This relation is not expressible in a relational refinement type system, since the two programs have different types. But they still have similar structure, and we wish to exploit this to find a proof. First we notice that both programs are recursive functions, so we apply a rule for this case, [LETREC]. This gives us the inductive hypothesis that the relation above holds for natural numbers strictly smaller than n, and the proof obligation that, for the following two programs

$$\text{case } n \text{ of } 0 \mapsto 1; S \mapsto \lambda x.(S\ x) * (f_1\ x)$$
$$\lambda acc.\text{case } n \text{ of } 0 \mapsto acc; S \mapsto \lambda x.f_2\ x\ ((S\ x) * acc)$$

the result of the first one, times a, is equal to the result calling the second one with argument a, that is,

$$\forall a. a * \mathbf{r}_1 = \mathbf{r}_2\ a$$

But now we have a pair of programs that have not only different types, but also different top-level term former: the first one has a case-analysis, while the second one has an abstraction. We now apply a one-sided rule [ABS-R] that takes care of the abstraction on the right and ignores the term on the left, leaving a pair of programs

$$\text{case } n \text{ of } 0 \mapsto 1; S \mapsto \lambda x.(S\ x) * (f_1\ x)$$
$$\text{case } n \text{ of } 0 \mapsto acc; S \mapsto \lambda x.f_2\ x\ ((S\ x) * a)$$

and a proof obligation

$$a * \mathbf{r}_1 = \mathbf{r}_2$$

Finally we have two programs with the same type and the same structure. We can do synchronous reasoning, using the fact that both programs need to take the same branch. In the 0 branch, its trivial to check that $a*1 = a$. In the successor branch we can instantiate the inductive hypothesis, since the bound variable x is the predecessor of n, and thus strictly smaller. This concludes the proof.

4 Higher-Order Logic

Higher-Order Logic is a predicate logic over simply-typed, higher-order terms. Its rules are written in the style of natural deduction. More specifically, its assertions are formulae over typed terms, and are defined by the following grammar:

$$\phi ::= P(t_1,\ldots,t_n) \mid \top \mid \bot \mid \phi \wedge \phi \mid \phi \vee \phi \mid \phi \Rightarrow \phi \mid \forall x:\tau.\phi \mid \exists x:\tau.\phi$$

where P ranges over basic predicates, such as $=$, $\leq$, or $\mathsf{sorted}(l)$. As usual, we will often omit the types of bound variables, when they are clear from the context. We assume that predicates come equipped with an axiomatization. For instance, the predicate $\mathsf{All}(l, \lambda x.\phi)$ is defined to capture lists whose elements e all satisfy $\phi[e/x]$. This can be defined axiomatically for a given ϕ with a free variable x (which we denote with and overloaded λ symbol):

$$\mathrm{All}([], \lambda x.\phi) \qquad\qquad \forall h t.\, \mathrm{All}(t, \lambda x.\phi) \Rightarrow \phi[h/x] \Rightarrow \mathrm{All}(h :: t, \lambda x.\phi)$$

The notation $\lambda x.\phi$, used for simplicity, can be made formal by introducing a type for propositions—adding such a type is straightforward and orthogonal to our work; another alternative would be to use axiom schemes.

We define well-typed assertions using a judgment of the form $\Gamma \vdash \phi$. This judgment has the expected, straightforward rules. A HOL (inference) judgment is then of the form $\Gamma \mid \Psi \vdash \phi$, where Γ is a simply typed context, Ψ is a set of assertions, and ϕ is an assertion, and such that $\Gamma \vdash \psi$ for every $\psi \in \Psi$, and $\Gamma \vdash \phi$. The rules of this judgment are given in Figure 1, where the notation $\phi[t/x]$ denotes the (capture-free) substitution of x by t in ϕ. In addition to the usual rules for equality, implication and universal quantification, there are rules for inductive types (only the rules for lists are shown; similar rules exist for booleans and natural numbers): the rule [LIST] models the induction principle for lists; the rules [NC] and [CONS] formalize injectivity and non-overlap of constructors. A rule for strong induction [SLIST] can be considered as well, and is in fact derivable from simple induction.

Higher-Order Logic inherits a set-theoretical interpretation from its underlying simply-typed λ-calculus. We assume given for each predicate P an interpretation $[\![P]\!]$ which is compatible with the type of P and its axioms. The interpretation of assertions is then defined in the usual way. Specifically, the interpretation $(\!|\phi|\!)_\rho$ of an assertion ϕ w.r.t. a valuation ρ includes the clauses:

$$(\!|P(t_1,\ldots,t_n)|\!)_\rho \triangleq ([\![t_1]\!]_\rho,\ldots,[\![t_n]\!]_\rho) \in [\![P]\!] \qquad (\!|\top|\!)_\rho \triangleq \tilde{\top} \qquad (\!|\bot|\!)_\rho \triangleq \tilde{\bot}$$

$$(\!|\phi_1 \wedge \phi_2|\!)_\rho \triangleq (\!|\phi_1|\!)_\rho \mathrel{\tilde{\wedge}} (\!|\phi_2|\!)_\rho \qquad (\!|\phi_1 \Rightarrow \phi_2|\!)_\rho \triangleq (\!|\phi_1|\!)_\rho \mathrel{\tilde{\Rightarrow}} (\!|\phi_2|\!)_\rho$$

$$(\!|\forall x:\tau.\phi|\!)_\rho \triangleq \tilde{\forall} v.\, v \in [\![\tau]\!] \mathrel{\tilde{\Rightarrow}} (\!|\phi|\!)_{\rho[v/x]}$$

$$\dfrac{\phi \in \Psi}{\Gamma \mid \Psi \vdash \phi}\ \textsf{AX} \qquad \dfrac{\Gamma \vdash t : \tau \quad \Gamma \vdash t' : \tau \quad t =_{\beta\iota\mu} t'}{\Gamma \mid \Psi \vdash t = t'}\ \textsf{CONV}$$

$$\dfrac{\Gamma \mid \Psi \vdash \phi[t/x] \quad \Gamma \mid \Psi \vdash t = u}{\Gamma \mid \Psi \vdash \phi[u/x]}\ \textsf{SUBST} \qquad \dfrac{\Gamma \mid \Psi, \psi \vdash \phi}{\Gamma \mid \Psi \vdash \psi \Rightarrow \phi}\ \Rightarrow_{\textsf{I}}$$

$$\dfrac{\Gamma \mid \Psi \vdash \psi \Rightarrow \phi \quad \Gamma \mid \Psi \vdash \psi}{\Gamma \mid \Psi \vdash \phi}\ \Rightarrow_{\textsf{E}} \qquad \dfrac{\Gamma, x : \sigma \mid \Psi \vdash \phi}{\Gamma \mid \Psi \vdash \forall x : \sigma . \phi}\ \forall_{\textsf{I}} \qquad \dfrac{\Gamma \mid \Psi \vdash \forall x : \sigma . \phi \quad \Gamma \vdash t : \sigma}{\Gamma \mid \Psi \vdash \phi[t/x]}\ \forall_{\textsf{E}}$$

$$\dfrac{}{\Gamma \mid \Psi \vdash \top}\ \top_{\textsf{I}} \qquad \dfrac{\Gamma \mid \Psi \vdash \bot \quad \Gamma \vdash \phi}{\Gamma \mid \Psi \vdash \phi}\ \bot_{\textsf{E}}$$

$$\dfrac{\Gamma \mid \Psi \vdash \phi[[]/l] \quad \Gamma, h : \tau, t : \textsf{list}_\tau \mid \Psi, \phi \vdash \phi[h :: t/t]}{\Gamma \mid \Psi \vdash \forall t : \textsf{list}_\sigma . \phi}\ \textsf{LIST} \qquad \dfrac{\Gamma \vdash h :: t : \textsf{list}_\tau}{\Gamma \mid \emptyset \vdash [] \neq h :: t}\ \textsf{NC}$$

$$\dfrac{\Gamma \mid \Psi \vdash t_1 :: t_2 = t'_1 :: t'_2}{\Gamma \mid \Psi \vdash t_i = t'_i}\ \textsf{CONS}_{\textsf{i}} \qquad \dfrac{\Gamma, t : \textsf{list}_\tau \mid \Psi, \forall u : \textsf{list}_\tau . |u| < |t| \Rightarrow \phi[u/t] \vdash \phi}{\Gamma \mid \Psi \vdash \forall t : \textsf{list}_\tau . \phi}\ \textsf{SLIST}$$

Fig. 1. Selected rules for HOL

where the tilde ($\sim$) is used to distinguish between HOL connectives and meta-level connectives. Higher-order logic is sound with respect to this semantics.

Theorem 2 (*Soundness of set-theoretical semantics*)
If $\Gamma \mid \Psi \vdash \phi$, then for every valuation $\rho \models \Gamma$, $\bigwedge_{\psi \in \Psi} (\!|\psi|\!)_\rho$ implies $(\!|\phi|\!)_\rho$.

In particular, HOL is consistent, i.e. there is no derivation of $\Gamma \mid \emptyset \vdash \bot$ for any Γ.

5 Unary Higher-Order Logic

As a stepping stone towards Relational Higher-Order Logic, we define Unary Higher-Order Logic (UHOL). UHOL retains the flavor of refinement types, but dissociates typing from assertions; judgments of UHOL are of the form:

$$\Gamma \mid \Psi \vdash t : \tau \mid \phi$$

where a distinguished variable $\mathbf{r}$, which doesn't appear in Γ, may appear free in ϕ as a synonym of t. A judgment is well-formed if t has type τ, Ψ is a valid set of assertions in the context Γ, and ϕ is a valid assertion in the context $\Gamma, \mathbf{r} : \tau$. Not only is UHOL a logical system in its own right, but it is also a key component for deriving one-sided rules in RHOL (Section 6). When reasoning about two terms with different structures, RHOL relies on UHOL to express proof obligations that affect only one of the two terms.

Figure 2 presents selected typing rules. The [$\textsf{ABS}$] rule allows proving formulas that refer to λ-abstractions, expressing that if the argument satisfies a precondition ϕ', then the result satisfies a postcondition ϕ. The [$\textsf{APP}$] rule, dually, proves a condition ϕ on an application $t\ u$ provided that the argument u satisfies the precondition ϕ' of the function t. The motivation behind the substitution $[\mathbf{r}\ x/\mathbf{r}]$ in both rules is that the type of $\mathbf{r}$ changes from the premise to the conclusion, and when $\mathbf{r}$ is bound to

ZU064-05-FPR main 16 August 2020 22:41

an arrow type, it needs to appear on the left of an application construct to recover its original type.

The [VAR] rule introduces a variable from the context with a formula proven in HOL. Rules for constants (e.g. [NIL]) work in the same way. Rule [CONS] proves a formula ϕ for a non-empty list, provided that ϕ on the entire list is a logical consequence (in HOL) of some conditions ϕ', ϕ'' on its head and its tail, respectively. Rule [PAIR] allows the construction of judgments about pairs in a similar manner. The rules [PROJ_i] for $i = 1, 2$ establish judgments about the projections of a pair. The rule [SUB] (subsumption) allows strengthening the assumed assertions Ψ and weakening the concluding assertion ϕ. It generates a HOL proof obligation. The rule [CASE] can be used for a case analysis over the constructor of a term. Finally, the rule [LETREC] supports inductive reasoning about recursive function. Recall that the domain of a recursive definition is an inductive type, for which a natural notion of size exists. If, assuming that a proposition holds for all elements smaller than the argument, we can prove that the proposition holds for the body, then the proposition must hold for the function as well. Furthermore, we require that the function we are verifying satisfies the predicate $\mathcal{D}ef(f, x, t)$, as was the case in HOL. The induction is performed over the $<$ order, which varies depending on the type of the argument.

We now discuss the main meta-theoretic results of UHOL. The following result establishes that every HOL judgment can be proven in UHOL and vice versa.

Theorem 3 (*Equivalence with HOL*)
For every context Γ, simple type σ, term t, set of assertions Ψ and assertion ϕ, the following are equivalent:

- $\Gamma \mid \Psi \vdash t : \sigma \mid \phi$
- $\Gamma \mid \Psi \vdash \phi[t/\mathbf{r}]$

The forward implication follows by induction on the derivation of $\Gamma \mid \Psi \vdash t : \sigma \mid \phi$. The reverse implication is immediate from the rule [SUB] and the observation that $\Gamma \mid \Psi \vdash t : \sigma \mid \top$ whenever t is a term of type σ.

We lift the HOL semantics to UHOL. Terms, types and formulas are interpreted as before. The following corollary states the soundness of UHOL.

Corollary 4 (*Set-theoretical soundness and consistency*)
Let t be a term such that $\Gamma \mid \Psi \vdash t : \sigma \mid \phi$ and ρ a valuation such that $\rho \models \Gamma$. If $\bigwedge_{\psi \in \Psi} (\!|\psi|\!)_\rho$, then $(\!|\phi|\!)_{\rho[(\!|t|\!)_\rho/\mathbf{r}]}$. In particular, there is no proof of the judgment $\Gamma \mid \emptyset \vdash t : \sigma \mid \bot$ for any Γ, t and σ.

Next, we prove subject conversion for UHOL. This result follows immediately from Theorem 3 and subject conversion of HOL, which is itself a direct consequence of the [CONV] and [SUBST] rules.

Corollary 5 (*Subject conversion*)
Assume that $t =_{\beta\iota\mu} t'$ and $\Gamma \mid \Psi \vdash t : \sigma \mid \phi$. Then $\Gamma \mid \Psi \vdash t' : \sigma \mid \phi$.

$$\frac{\Gamma \vdash x : \sigma \qquad \Gamma \mid \Psi \vdash \phi[x/\mathbf{r}]}{\Gamma \mid \Psi \vdash x : \sigma \mid \phi}\ \textsf{VAR} \qquad \frac{\Gamma, x : \tau \mid \Psi, \phi' \vdash t : \sigma \mid \phi}{\Gamma \mid \Psi \vdash \lambda x : \tau . t : \tau \to \sigma \mid \forall x. \phi' \Rightarrow \phi[\mathbf{r}\ x/\mathbf{r}]}\ \textsf{ABS}$$

$$\frac{\Gamma \mid \Psi \vdash t : \tau \to \sigma \mid \forall x. \phi'[x/\mathbf{r}] \Rightarrow \phi[\mathbf{r}\ x/\mathbf{r}] \qquad \Gamma \mid \Psi \vdash u : \tau \mid \phi'}{\Gamma \mid \Psi \vdash t\ u : \sigma \mid \phi[u/x]}\ \textsf{APP}$$

$$\frac{\Gamma \mid \Psi \vdash_{\textsf{HOL}} \phi[[]/\mathbf{r}]}{\Gamma \mid \Psi \vdash [] : \textsf{list}_\sigma \mid \phi}\ \textsf{NIL} \qquad \frac{\begin{array}{c}\Gamma \mid \Psi \vdash h : \sigma \mid \phi' \qquad \Gamma \mid \Psi \vdash t : \textsf{list}_\sigma \mid \phi'' \\ \Gamma \mid \Psi \vdash_{\textsf{HOL}} \forall xy. \phi'[x/\mathbf{r}] \Rightarrow \phi''[y/\mathbf{r}] \Rightarrow \phi[x :: y/\mathbf{r}]\end{array}}{\Gamma \mid \Psi \vdash h :: t : \textsf{list}_\sigma \mid \phi}\ \textsf{CONS}$$

$$\frac{\Gamma \mid \Psi \vdash t : \sigma \times \tau \mid \phi[\pi_i(\mathbf{r})/\mathbf{r}]}{\Gamma \mid \Psi \vdash \pi_i(t) : \sigma \mid \phi}\ \textsf{PROJ}_\textsf{i}$$

$$\frac{\begin{array}{c}\Gamma \mid \Psi \vdash t : \sigma \mid \phi' \qquad \Gamma \mid \Psi \vdash u : \tau \mid \phi'' \\ \Gamma \mid \Psi \vdash_{\textsf{HOL}} \forall xy. \phi'[x/\mathbf{r}] \Rightarrow \phi''[y/\mathbf{r}] \Rightarrow \phi[\langle x, y\rangle/\mathbf{r}]\end{array}}{\Gamma \mid \Psi \vdash \langle t, u\rangle : \sigma \times \tau \mid \phi}\ \textsf{PAIR}$$

$$\frac{\Gamma \mid \Psi \vdash t : \sigma \mid \phi' \quad \Gamma \mid \Psi \vdash_{\textsf{HOL}} \phi'[t/\mathbf{r}] \Rightarrow \phi[t/\mathbf{r}]}{\Gamma \mid \Psi \vdash t : \sigma \mid \phi}\ \textsf{SUB}$$

$$\frac{\begin{array}{c}\Gamma \vdash l : \textsf{list}_\tau \\ \Gamma \mid \Psi, l = [] \vdash u : \sigma \mid \phi \\ \Gamma \mid \Psi \vdash v : \tau \to \textsf{list}_\tau \to \sigma \mid \forall ht. l = h :: t \Rightarrow \phi[\mathbf{r}\ h\ t/\mathbf{r}]\end{array}}{\Gamma \mid \Psi \vdash \text{case } l \text{ of } [] \mapsto u; \_ :: \_ \mapsto v : \sigma \mid \phi}\ \textsf{LISTCASE}$$

$$\frac{\begin{array}{c}\mathcal{D}ef(f, x, t) \\ \Gamma, x : I, f : I \to \sigma \mid \Psi, \phi', \forall m. |m| < |x| \Rightarrow \phi'[m/x] \Rightarrow \phi[m/x][f\ m/\mathbf{r}] \vdash e : \sigma \mid \phi\end{array}}{\Gamma \mid \Psi \vdash \text{letrec } f\ x\ = t : I \to \sigma \mid \forall x. \phi' \Rightarrow \phi[\mathbf{r}\ x/\mathbf{r}]}\ \textsf{LETREC}$$

$$\text{where } I \in \{\mathbb{N}, \textsf{list}_\tau\}$$

Fig. 2. Unary Higher-Order Logic rules

6 Relational Higher-Order Logic

Relational Higher-Order Logic (RHOL) extends UHOL's separation of assertions and types to a relational setting. Formally, RHOL is a relational type system which manipulates judgments of the form

$$\Gamma \mid \Psi \vdash t_1 : \tau_1 \sim t_2 : \tau_2 \mid \phi$$

which combine a typing judgment for a pair of PCF terms and permit reasoning about the relation between them. The judgment means that t_1 and t_2 respectively have types τ_1 and τ_2 in Γ and that t_1, t_2 are related by the assertion ϕ. Well-formedness of the judgment requires Ψ to be a valid set of assertions in Γ and ϕ to be a valid assertion in $\Gamma, \mathbf{r}_1 : \tau_1, \mathbf{r}_2 : \tau_2$, where the special variables $\mathbf{r}_1$ and $\mathbf{r}_2$ are used as synonyms for t_1 and t_2 in ϕ.

6.1 Proof Rules

The type system combines two-sided rules (Figure 3), which apply when the two terms have the same top-level constructors and one-sided rules (Figure 5), which analyze either one of the two terms. For instance, the [APP] rule applies when the two terms are applications, and requires that the functions t_1 and t_2 relate and the arguments u_1 and u_2 relate. Specifically, t_1 and t_2 must map values related by ϕ' to values related by ϕ, and u_1 and u_2 must be related by ϕ'. The [ABS] rule is dual. The [PAIR] rule requires that the left and right components of a pair relate independently (a stronger rule is discussed at the end of this section). The [PROJ$_i$] rules require in their premise an assertion that only refers to the first or the second component of the pair. The rules for lists require that the two lists are either both empty, or both non-empty. The rule [CONS] requires that the two heads and the two tails relate independently. The [CASE] rule derives judgments about two case constructs when the terms over which the matching happens reduce to the same branch (i.e. have the same constructor) on both sides.

In contrast, one-sided typing rules only analyze one term; therefore, they come in two flavors: left rules (shown in Figure 5) and right rules (omitted but similar). Rule [ABS-L] considers the case where the left term is a λ-abstraction, and requires the body of the abstraction to be related to the right term u_2 whenever the argument on the left side satisfies a non-relational assertion ϕ'. Dually, rule [APP-L] considers the case where the left term is of the form $t_1\ u_1$, and t_1 is related to the right term u_2; specifically, t_1 should map every value satisfying ϕ' to a value satisfying ϕ. Moreover, u_1 should satisfy ϕ' in UHOL (not RHOL, since ϕ' is a non-relational assertion). One-sided rules for pairs and lists follow a similar pattern.

In addition, RHOL has structural rules (Figure 4). The rule [SUB] can be used for weakening the conclusion; the ensuing side-condition is discharged in HOL. Other structural rules assimilate rules of HOL. For instance, if we can prove two different assertions for the same terms we can prove the conjunction of the assertions ([$\wedge_I$]). Other logical connectives have similar rules. Finally, the rule [UHOL-L] (and a dual rule [UHOL-R]) allow falling back to UHOL in a RHOL proof.

Rules [LETREC] and [LETREC-L] introduce recursive function definitions (Figure 6). These rules allow for a style of reasoning very similar to strong induction. If, assuming that the function's specification holds for all smaller arguments, we can prove that the functions specification holds, then the specification must hold for all arguments. We require that the two functions we are relating satisfy the predicates $\mathcal{D}ef(f_i, x_i, t_i)$, as was the case in HOL and UHOL. The induction is performed over the simultaneous order $(a,b) < (c,d)$, which holds whenever both $a \leq b$ and $c \leq d$, and at least one of the inequalities is strict.

6.2 Discussion

6.2.1 Management of the typing context

In a relational judgment $\Gamma \mid \Psi \vdash t_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi$, the context Γ contains variables that can appear free in both t_1 and t_2. This forces us to add additional premises to

$$\frac{\Gamma, x_1 : \tau_1, x_2 : \tau_2 \mid \Psi, \phi' \vdash t_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi \qquad x_1 \notin \mathsf{FV}(t_2) \qquad x_2 \notin \mathsf{FV}(t_1)}{\Gamma \mid \Psi \vdash \lambda x_1 : \tau_1 . t_1 : \tau_1 \to \sigma_1 \sim \lambda x_2 : \tau_2 . t_2 : \tau_2 \to \sigma_2 \mid \forall x_1, x_2 . \phi' \Rightarrow \phi[\mathbf{r}_1\ x_1/\mathbf{r}_1][\mathbf{r}_2\ x_2/\mathbf{r}_2]} \ \mathsf{ABS}$$

$$\frac{\begin{array}{c}\Gamma \mid \Psi \vdash t_1 : \tau_1 \to \sigma_1 \sim t_2 : \tau_2 \to \sigma_2 \mid \forall x_1, x_2 . \phi'[x_1/\mathbf{r}_1][x_2/\mathbf{r}_2] \Rightarrow \phi[\mathbf{r}_1\ x_1/\mathbf{r}_1][\mathbf{r}_2\ x_2/\mathbf{r}_2] \\ \Gamma \mid \Psi \vdash u_1 : \tau_1 \sim u_2 : \tau_2 \mid \phi'\end{array}}{\Gamma \mid \Psi \vdash t_1 u_1 : \sigma_1 \sim t_2 u_2 : \sigma_2 \mid \phi[u_1/x_1][u_2/x_2]} \ \mathsf{APP}$$

$$\frac{\Gamma \vdash x_1 : \sigma_1 \quad \Gamma \vdash x_2 : \sigma_2 \quad \Gamma \mid \Psi \vdash \phi[x_1/\mathbf{r}_1][x_2/\mathbf{r}_2]}{\Gamma \mid \Psi \vdash x_1 : \sigma_1 \sim x_2 : \sigma_2 \mid \phi} \ \mathsf{VAR}$$

$$\frac{\Gamma \mid \Psi \vdash_{\mathsf{HOL}} \phi[\mathsf{tt}/\mathbf{r}_1][\mathsf{tt}/\mathbf{r}_2]}{\Gamma \mid \Psi \vdash \mathsf{tt} : \mathbb{B} \sim \mathsf{tt} : \mathbb{B} \mid \phi} \ \mathsf{TRUE} \qquad \frac{\Gamma \mid \Psi \vdash_{\mathsf{HOL}} \phi[[]/\mathbf{r}_1][[]/\mathbf{r}_2]}{\Gamma \mid \Psi \vdash [] : \mathsf{list}_{\sigma_1} \sim [] : \mathsf{list}_{\sigma_2} \mid \phi} \ \mathsf{NIL}$$

$$\frac{\begin{array}{c}\Gamma \mid \Psi \vdash h_1 : \sigma_1 \sim h_2 : \sigma_2 \mid \phi' \qquad\qquad \Gamma \mid \Psi \vdash t_1 : \mathsf{list}_{\sigma_1} \sim t_2 : \mathsf{list}_{\sigma_2} \mid \phi'' \\ \Gamma \mid \Psi \vdash_{\mathsf{HOL}} \forall x_1 x_2 y_1 y_2 . \phi'[x_1/\mathbf{r}_1][x_2/\mathbf{r}_2] \Rightarrow \phi''[y_1/\mathbf{r}_1][y_2/\mathbf{r}_2] \Rightarrow \phi[x_1 :: y_1/\mathbf{r}_1][x_2 :: y_2/\mathbf{r}_2]\end{array}}{\Gamma \mid \Psi \vdash h_1 :: t_1 : \mathsf{list}_{\sigma_1} \sim h_2 :: t_2 : \mathsf{list}_{\sigma_2} \mid \phi} \ \mathsf{CONS}$$

$$\frac{\begin{array}{c}\Gamma \mid \Psi \vdash l_1 : \mathsf{list}_{\tau_1} \sim l_2 : \mathsf{list}_{\tau_2} \mid \mathbf{r}_1 = [] \Leftrightarrow \mathbf{r}_2 = [] \\ \Gamma \mid \Psi, l_1 = [], l_2 = [] \vdash u_1 : \sigma_1 \sim u_2 : \sigma_2 \mid \phi \\ \Gamma \mid \Psi \vdash v_1 : \tau_1 \to \mathsf{list}_{\tau_1} \to \sigma_1 \sim v_2 : \tau_2 \to \mathsf{list}_{\tau_2} \to \sigma_2 \mid \\ \forall h_1 h_2 t_1 t_2 . l_1 = h_1 :: t_1 \Rightarrow l_2 = h_2 :: t_2 \Rightarrow \phi[\mathbf{r}_1\ h_1\ t_1/\mathbf{r}_1][\mathbf{r}_2\ h_2\ t_2/\mathbf{r}_2]\end{array}}{\Gamma \mid \Psi \vdash \mathrm{case}\ l_1\ \mathrm{of}\ [] \mapsto u_1; \_ :: \_ \mapsto v_1 : \sigma_1 \sim \mathrm{case}\ l_2\ \mathrm{of}\ [] \mapsto u_2; \_ :: \_ \mapsto v_2 : \sigma_2 \mid \phi} \ \mathsf{LISTCASE}$$

$$\frac{\begin{array}{c}\Gamma \mid \Psi \vdash t_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi' \qquad\qquad \Gamma \mid \Psi \vdash u_1 : \tau_1 \sim u_2 : \tau_2 \mid \phi'' \\ \Gamma \mid \Psi \vdash_{\mathsf{HOL}} \forall x_1 x_2 y_1 y_2 . \phi'[x_1/\mathbf{r}_1][x_2/\mathbf{r}_2] \Rightarrow \phi''[y_1/\mathbf{r}_1][y_2/\mathbf{r}_2] \Rightarrow \phi[\langle x_1, y_1 \rangle/\mathbf{r}_1][\langle x_2, y_2 \rangle/\mathbf{r}_2]\end{array}}{\Gamma \mid \Psi \vdash \langle t_1, u_1 \rangle : \sigma_1 \times \tau_1 \sim \langle t_2, u_2 \rangle : \sigma_2 \times \tau_2 \mid \phi} \ \mathsf{PAIR}$$

$$\frac{\Gamma \mid \Psi \vdash t_1 : \sigma_1 \times \tau_1 \sim t_2 : \sigma_2 \times \tau_2 \mid \phi[\pi_i(\mathbf{r}_1)/\mathbf{r}_1][\pi_i(\mathbf{r}_2)/\mathbf{r}_2]}{\Gamma \mid \Psi \vdash \pi_i(t_1) : \sigma_1 \sim \pi_i(t_2) : \sigma_2 \mid \phi} \ \mathsf{PROJ_i}$$

Fig. 3. Two-sided rules

$$\frac{\Gamma \mid \Psi \vdash t_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi' \quad \Gamma \mid \Psi \vdash_{\mathsf{HOL}} \phi'[t_1/\mathbf{r}_1][t_2/\mathbf{r}_2] \Rightarrow \phi[t_1/\mathbf{r}_1][t_2/\mathbf{r}_2]}{\Gamma \mid \Psi \vdash t_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi} \ \mathsf{SUB}$$

$$\frac{\Gamma \mid \Psi \vdash t_1 : \sigma_2 \sim t_2 : \sigma_2 \mid \phi \quad \Gamma \mid \Psi \vdash t_1 : \sigma_2 \sim t_2 : \sigma_2 \mid \phi'}{\Gamma \mid \Psi \vdash t_1 : \sigma_2 \sim t_2 : \sigma_2 \mid \phi \wedge \phi'} \ \wedge_{\mathsf{I}}$$

$$\frac{\Gamma \mid \Psi, \phi'[t_1/\mathbf{r}_1][t_2/\mathbf{r}_2] \vdash t_1 : \sigma_2 \sim t_2 : \sigma_2 \mid \phi}{\Gamma \mid \Psi \vdash t_1 : \sigma_2 \sim t_2 : \sigma_2 \mid \phi' \Rightarrow \phi} \ \Rightarrow_{\mathsf{I}}$$

$$\frac{\Gamma \mid \Psi \vdash t_1 : \sigma_1 \mid \phi[\mathbf{r}/\mathbf{r}_1][t_2/\mathbf{r}_2] \qquad \Gamma \vdash t_2 : \sigma_2}{\Gamma \mid \Psi \vdash t_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi} \ \mathsf{UHOL-L}$$

Fig. 4. Structural rules

$$\frac{\Gamma, x_1 : \tau_1 \mid \Psi, \phi' \vdash t_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi \qquad x_1 \notin \mathsf{FV}(t_2)}{\Gamma \mid \Psi \vdash \lambda x_1 : \tau_1 . t_1 : \tau_1 \to \sigma_1 \sim t_2 : \sigma_2 \mid \forall x_1 . \phi' \Rightarrow \phi[\mathbf{r}_1\ x_1/\mathbf{r}_1]}\ \textsf{ABS}{-}\textsf{L}$$

$$\frac{\begin{array}{c}\Gamma \mid \Psi \vdash t_1 : \tau_1 \to \sigma_1 \sim u_2 : \sigma_2 \mid \forall x_1 . \phi'[x_1/\mathbf{r}_1] \Rightarrow \phi[\mathbf{r}_1\ x_1/\mathbf{r}_1] \\ \Gamma \mid \Psi \vdash u_1 : \sigma_1 \mid \phi'\end{array}}{\Gamma \mid \Psi \vdash t_1 u_1 : \sigma_1 \sim u_2 : \sigma_2 \mid \phi[u_1/x_1]}\ \textsf{APP}{-}\textsf{L}$$

$$\frac{\phi[x_1/\mathbf{r}_1] \in \Psi \quad \mathbf{r}_2 \notin FV(\phi) \quad \Gamma \vdash t_2 : \sigma_2}{\Gamma \mid \Psi \vdash x_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi}\ \textsf{VAR}{-}\textsf{L}$$

$$\frac{\Gamma \mid \Psi \vdash \phi[[]/\mathbf{r}_1][t_2/\mathbf{r}_2] \quad \Gamma \vdash t_2 : \sigma_2}{\Gamma \mid \Psi \vdash [] : \mathsf{list}_{\sigma_1} \sim t_2 : \sigma_2 \mid \phi}\ \textsf{NIL}{-}\textsf{L}$$

$$\frac{\begin{array}{c}\Gamma \mid \Psi \vdash h_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi' \qquad\qquad \Gamma \mid \Psi \vdash t_1 : \mathsf{list}_{\sigma_1} \sim t_2 : \sigma_2 \mid \phi'' \\ \Gamma \mid \Psi \vdash_{\mathsf{HOL}} \forall x_1 x_2 y_1 . \phi'[x_1/\mathbf{r}_1][x_2/\mathbf{r}_2] \Rightarrow \phi''[y_1/\mathbf{r}_1][x_2/\mathbf{r}_2] \Rightarrow \phi[x_1 :: y_1/\mathbf{r}_1][x_2/\mathbf{r}_2]\end{array}}{\Gamma \mid \Psi \vdash h_1 :: t_1 : \mathsf{list}_{\sigma_1} \sim t_2 : \sigma_2 \mid \phi}\ \textsf{CONS}{-}\textsf{L}$$

$$\frac{\begin{array}{c}\Gamma \vdash t_1 : \mathsf{list}_\tau \\ \Gamma \mid \Psi, t_1 = [] \vdash u_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi \\ \Gamma \mid \Psi \vdash v_1 : \tau \to \mathsf{list}_\tau \to \sigma_1 \sim t_2 : \sigma_2 \mid \forall h_1 l_1 . t_1 = h_1 :: l_1 \Rightarrow \phi[\mathbf{r}_1\ h_1\ l_1/\mathbf{r}_1]\end{array}}{\Gamma \mid \Psi \vdash \text{case } t_1 \text{ of } [] \mapsto u_1; \_ :: \_ \mapsto v_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi}\ \textsf{LISTCASE}-\textsf{L}$$

$$\frac{\begin{array}{c}\Gamma \mid \Psi \vdash t_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi' \qquad\qquad \Gamma \mid \Psi \vdash u_1 : \tau_1 \sim t_2 : \sigma_2 \mid \phi'' \\ \Gamma \mid \Psi \vdash_{\mathsf{HOL}} \forall x_1 x_2 y_1 . \phi'[x_1/\mathbf{r}_1][x_2/\mathbf{r}_2] \Rightarrow \phi''[y_1/\mathbf{r}_1][x_2/\mathbf{r}_2] \Rightarrow \phi[\langle x_1, y_1\rangle/\mathbf{r}_1][x_2/\mathbf{r}_2]\end{array}}{\Gamma \mid \Psi \vdash \langle t_1, u_1 \rangle : \sigma_1 \times \tau_1 \sim t_2 : \sigma_2 \mid \phi}\ \textsf{PAIR}{-}\textsf{L}$$

$$\frac{\Gamma \mid \Psi \vdash t_1 : \sigma_1 \times \tau_1 \sim t_2 : \sigma_2 \mid \phi[\pi_1(\mathbf{r}_1)/\mathbf{r}_1]}{\Gamma \mid \Psi \vdash \pi_1(t_1) : \sigma_1 \sim t_2 : \sigma_2 \mid \phi}\ \textsf{PROJ}_1{-}\textsf{L}$$

Fig. 5. One-sided rules

$$\frac{\begin{array}{c}\mathcal{D}ef(f_1, x_1, t_1) \quad \mathcal{D}ef(f_2, x_2, t_2) \quad x_1, f_1 \notin \mathsf{FV}(t_2) \quad x_2, f_2 \notin \mathsf{FV}(t_1) \\ \Gamma, x_1 : I_1, x_2 : I_2, f_1 : I_1 \to \sigma, f_2 : I_2 \to \sigma_2 \mid \\ \Psi, \phi', \forall m_1 m_2 . (|m_1|, |m_2|) < (|x_1|, |x_2|) \Rightarrow \phi'[m_1/x_1][m_2/x_2] \Rightarrow \\ \phi[m_1/x_1][m_2/x_2][f_1\ m_1/\mathbf{r}_1][f_2\ m_2/\mathbf{r}_2] \vdash t_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi\end{array}}{\Gamma \mid \Psi \vdash \begin{array}{c}\text{letrec } f_1\ x_1\ = t_1 : I_1 \to \sigma_2 \sim \\ \text{letrec } f_2\ x_2\ = t_2 : I_2 \to \sigma_2\end{array} \mid \forall x_1 x_2 . \phi' \Rightarrow \phi[\mathbf{r}_1\ x_1/\mathbf{r}_1][\mathbf{r}_2\ x_2/\mathbf{r}_2]}\ \textsf{LETREC}$$

$$\frac{\begin{array}{c}\mathcal{D}ef(f_1, x_1, t_1) \quad x_1, f_1 \notin \mathsf{FV}(t_2) \\ \Gamma, x_1 : I_1, f_1 : I_1 \to \sigma \mid \Psi, \phi', \forall m_1 . |m_1| < |x_1| \Rightarrow \phi'[m_1/x_1] \Rightarrow \\ \phi[m_1/x_1][f_1\ m_1/\mathbf{r}_1][t_2/\mathbf{r}_2] \vdash t_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi\end{array}}{\Gamma \mid \Psi \vdash \text{letrec } f_1\ x_1\ = t_1 : I_1 \to \sigma_2 \sim t_2 : \sigma_2 \mid \forall x_1 . \phi' \Rightarrow \phi[\mathbf{r}_1\ x_1/\mathbf{r}_1]}\ \textsf{LETREC}-\textsf{L}$$

where $I_1, I_2 \in \{\mathbb{N}, \mathsf{list}_\tau\}$

Fig. 6. Recursion rules

$$\frac{\Gamma \mid \Psi \vdash t_1 : \tau_1 \to \sigma_1 \sim t_2 : \tau_2 \to \sigma_2 \mid \phi[\mathbf{r}_1\ u_1/\mathbf{r}_1][\mathbf{r}_2\ u_2/\mathbf{r}_2]}{\Gamma \mid \Psi \vdash t_1\ u_1 : \sigma_1 \sim t_2\ u_2 : \sigma_2 \mid \phi}\ \textsf{APP}-\textsf{FUN}$$

$$\frac{\Gamma \mid \Psi \vdash u_1 : \tau_1 \sim u_2 : \tau_2 \mid \phi[t_1\ \mathbf{r}_1/\mathbf{r}_1][t_2\ \mathbf{r}_2/\mathbf{r}_2]}{\Gamma \mid \Psi \vdash t_1\ u_1 : \sigma_1 \sim t_2\ u_2 : \sigma_2 \mid \phi}\ \textsf{APP}-\textsf{ARG}$$

$$\frac{\Gamma \mid \Psi \vdash t_1 : \tau_1 \sim t_2 : \tau_2 \mid \phi[\langle \mathbf{r}_1, u_1\rangle/\mathbf{r}_1][\langle \mathbf{r}_2, u_2\rangle/\mathbf{r}_2]}{\Gamma \mid \Psi \vdash \langle t_1, u_1\rangle : \tau_1 \times \sigma_1 \sim \langle t_2, u_2\rangle : \tau_2 \times \sigma_2 \mid \phi}\ \textsf{PAIR}-\textsf{FST}$$

$$\frac{\Gamma \mid \Psi \vdash u_1 : \sigma_1 \sim u_2 : \sigma_2 \mid \phi[\langle t_1, \mathbf{r}_1\rangle/\mathbf{r}_1][\langle t_2, \mathbf{r}_2\rangle/\mathbf{r}_2]}{\Gamma \mid \Psi \vdash \langle t_1, u_1\rangle : \tau_1 \times \sigma_1 \sim \langle t_2, u_2\rangle : \tau_2 \times \sigma_2 \mid \phi}\ \textsf{PAIR}-\textsf{SND}$$

$$\frac{\begin{array}{c}\Gamma \mid \Psi \vdash t_1 : \mathsf{list}_{\tau_1} \sim t_2 : \mathsf{list}_{\tau_2} \mid \top \\ \Gamma \mid \Psi, t_1 = [], t_2 = [] \vdash u_1 : \sigma_1 \sim u_2 : \sigma_2 \mid \phi \\ \Gamma \mid \Psi, t_2 = [] \vdash v_1 : \tau_1 \to \mathsf{list}_{\tau_1} \to \sigma_1 \sim u_2 : \sigma_2 \mid \forall h_1 l_1 . t_1 = h_1 :: l_1 \Rightarrow \phi[\mathbf{r}_1\ h_1\ l_1/\mathbf{r}_1] \\ \Gamma \mid \Psi, t_1 = [] \vdash u_1 : \sigma_1 \sim v_2 : \tau_2 \to \mathsf{list}_{\tau_2} \to \sigma_2 \mid \forall h_2 l_2 . t_2 = h_2 :: l_2 \Rightarrow \phi[\mathbf{r}_2\ h_2\ l_2/\mathbf{r}_2] \\ \Gamma \mid \Psi \vdash v_1 : \tau_1 \to \mathsf{list}_{\tau_1} \to \sigma_1 \sim v_2 : \tau_2 \to \mathsf{list}_{\tau_2} \to \sigma_2 \mid \\ \forall h_1 h_2 l_1 l_2 . t_1 = h_1 :: l_1 \Rightarrow t_2 = h_2 :: l_2 \Rightarrow \phi[\mathbf{r}_1\ h_1\ l_1/\mathbf{r}_1][\mathbf{r}_2\ h_2\ l_2/\mathbf{r}_2]\end{array}}{\Gamma \mid \Psi \vdash \mathrm{case}\ t_1\ \mathrm{of}\ [] \mapsto u_1; \_ :: \_ \mapsto v_1 : \sigma_1 \sim \mathrm{case}\ t_2\ \mathrm{of}\ [] \mapsto u_2; \_ :: \_ \mapsto v_2 : \sigma_2 \mid \phi}\ \textsf{LLCASE}-\textsf{A}$$

Fig. 7. Some derived rules

the abstraction rules to ensure the terms in the conclusion remain well-typed. Note that we could have another sound (but less expressive) version of the [ABS] rule where we abstract over the same variable on both sides.

An alternative choice of presentation is to have separate contexts Γ_1 and Γ_2 for the terms on each side of the relational judgments, eliminating the need for the extra premises. However, there would be no other benefit to this alternative. For instance, UHOL premises as in [APP-L] would still need to use both contexts Γ_1 and Γ_2, since the logical context Ψ is typed under both.

6.2.2 Variants of the rules

Our choice of the rules is guided by the practical considerations of being able to verify examples easily, without specifically aiming for minimality or exhaustiveness. In fact, many of our rules can be derived from others, or reduced to a more elementary form. For instance:

- The structural rules to reason about logical connectives, such as $[\wedge_I]$, can be derived by induction on the length of derivations with the help of [SUB].
- The [VAR-L] (similarly, [NIL-L]) rule can be weakened, without affecting the strength of the system,

$$\frac{\phi[x_1/\mathbf{r}_1] \in \Psi \quad \mathbf{r}_2 \notin FV(\phi)}{\Gamma \mid \Psi \vdash x_1 : \sigma_1 \sim x_2 : \sigma_2 \mid \phi}\ \textsf{VAR}\text{—}\textsf{L}$$

- The premise of the [VAR] rule in Figure 3 (and similarly, [NIL]) can be changed to $\phi[x/\mathbf{r}] \in \Psi$. We can recover the original rule by one application of [SUB].

- The rules [APP-FUN] and [APP-ARG] in Figure 7 (adapted from (Ghani *et al.*, 2016b)) can be derived from the rule [APP]. To derive [APP-FUN], instantiate ϕ' to $\mathbf{r}_1 = u_1 \wedge \mathbf{r}_2 = u_2$ in [APP]. To derive [APP-ARG], we have to prove a trivial condition $\forall x_1 x_2 . \phi[t_1\ x_1/\mathbf{r}_1][t_2\ x_2/\mathbf{r}_2] \Rightarrow \phi[t_1\ x_1/\mathbf{r}_1][t_2\ x_2/\mathbf{r}_2]$ on t_1, t_2.
- The [PAIR-FST] and [PAIR-SND] rules in Figure 7 can be derived in a similar way. These rules overcome a limitation of the original [PAIR] rule, namely, that the relations for the two components of the pair must be independent. [PAIR-FST] and [PAIR-SND] allow relating, for instance, pairs of integers $\langle m_1, n_1 \rangle$ and $\langle m_2, n_2 \rangle$ such that $m_1 + n_1 = m_2 + n_2$.
- The [LLCASE-A] rule can be used to reason about case constructs when the terms over which we discriminate do not necessarily reduce to the same branch. It is equivalent to applying [LISTCASE-L] followed by [LISTCASE-R].

6.3 Meta-theory

RHOL retains the expressiveness of HOL, and is relatively complete with respect to it, as formalized in the following theorem.

Theorem 6 (*Equivalence with HOL*)
For every context Γ, simple types σ_1 and σ_2, terms t_1 and t_2, set of assertions Ψ and assertion ϕ, if $\Gamma \vdash t_1 : \sigma_1$ and $\Gamma \vdash t_2 : \sigma_2$, then the following are equivalent:

- $\Gamma \mid \Psi \vdash t_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi$
- $\Gamma \mid \Psi \vdash \phi[t_1/\mathbf{r}_1][t_2/\mathbf{r}_2]$

Proof
The easier direction is the reverse implication. To prove it, we can trivially apply [SUB] instantiating ϕ' as a tautology that matches the structure of the types. For instance, for a base type $\mathbb{N}$ we would use $\top$, for an arrow type $\mathbb{N} \to \mathbb{N}$ we would use $\forall x. \bot \Rightarrow \top$, and so on.

The forward implication follows by induction on the derivation of $\Gamma \mid \Psi \vdash t_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi$. We show here the cases for abstraction and application:

Case [ABS]. The rule is:

$$\frac{\Gamma, x_1 : \tau_1, x_2 : \tau_2 \mid \Psi, \phi' \vdash t_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi}{\Gamma \mid \Psi \vdash \lambda x_1 . t_1 : \tau_1 \to \sigma_1 \sim \lambda x_2 . t_2 : \tau_2 \to \sigma_2 \mid \forall x_1, x_2 . \phi' \Rightarrow \phi[\mathbf{r}_1\ x_1/\mathbf{r}_1][\mathbf{r}_2\ x_2/\mathbf{r}_2]}$$

By applying the induction hypothesis on the premise:
$\Gamma, x_1 : \tau_1, x_2 : \tau_2 \mid \Psi, \phi' \vdash \phi[t_1/\mathbf{r}_1][t_2/\mathbf{r}_2]$ (1)
By applying $[\Rightarrow_I]$ on (1):
$\Gamma, x_1 : \tau_1, x_2 : \tau_2 \mid \Psi \vdash \phi' \Rightarrow \phi[t_1/\mathbf{r}_1][t_2/\mathbf{r}_2]$ (2)
By applying $[\forall_I]$ twice on (2):
$\Gamma \mid \Psi \vdash \forall x_1 x_2 . \phi' \Rightarrow \phi[t_1/\mathbf{r}_1][t_2/\mathbf{r}_2]$ (3)
Finally, by applying CONV on (3):
$\Gamma \mid \Psi \vdash \forall x_1 x_2 . \phi' \Rightarrow \phi[(\lambda x_1 . t_1)\ x_1/\mathbf{r}_1][(\lambda x_2 . t_2)\ x_2/\mathbf{r}_2]$
The proof for [ABS-L] (and [ABS-R]) is analogous.

Case [APP]. The rule is

$$\frac{\begin{array}{c}\Gamma \mid \Psi \vdash t_1 : \tau_1 \to \sigma_1 \sim t_2 : \tau_2 \to \sigma_2 \mid \forall x_1, x_2 . \phi'[x_1/\mathbf{r}_1][x_2/\mathbf{r}_2] \Rightarrow \phi[\mathbf{r}_1\ x_1/\mathbf{r}_1][\mathbf{r}_2\ x_2/\mathbf{r}_2] \\ \Gamma \mid \Psi \vdash u_1 : \tau_1 \sim u_2 : \tau_2 \mid \phi'\end{array}}{\Gamma \mid \Psi \vdash t_1 u_1 : \sigma_1 \sim t_2 u_2 : \sigma_2 \mid \phi[u_1/x_1][u_2/x_2]}$$

By applying the induction hypothesis on the premises we have:
$\Gamma \mid \Psi \vdash \forall x_1 x_2 . \phi'[x_1/\mathbf{r}_1][x_2/\mathbf{r}_2] \Rightarrow \phi[t_1\ x_1/\mathbf{r}_1][t_2\ x_2/\mathbf{r}_2]$ (1)
and
$\Gamma \mid \Psi \vdash \phi'[u_1/\mathbf{r}_1][u_2/\mathbf{r}_2]$ (2)
By applying twice $[\forall_E]$ to (1) with u_1, u_2:
$\Gamma \mid \Psi \vdash \phi'[u_1/\mathbf{r}_1][u_2/\mathbf{r}_2] \Rightarrow \phi[t_1\ u_1/\mathbf{r}_1][t_2\ u_2/\mathbf{r}_2]$ (3)
and by applying $[\Rightarrow_E]$ to (3) and (2):
$\Gamma \mid \Psi \vdash \phi[t_1\ u_1/\mathbf{r}_1][t_2\ u_2/\mathbf{r}_2]$
The proof for [APP-L] (and APP-R) is analogous, and it uses the UHOL embedding (Theorem 3) for the premise about the argument. Proofs for [CONS] and [PAIR] also follow the same structure. □

This immediately entails soundness of RHOL, as formalized next.

Corollary 7 (*Set-theoretical soundness and consistency*)
Let t_1, t_2 be two terms such that $\Gamma \mid \Psi \vdash t_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi$ and ρ a valuation such that $\rho \models \Gamma$. If $\bigwedge_{\psi \in \Psi} (\!|\psi|\!)_\rho$, then $(\!|\phi|\!)_{\rho[(\!|t_1|\!)_\rho/\mathbf{r}_1],[(\!|t_2|\!)_\rho/\mathbf{r}_2]}$. In particular, there is no proof of the judgment $\Gamma \mid \emptyset \vdash t_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \bot$ for any Γ, t_1, t_2, σ_1 and σ_2.

The equivalence also entails subject conversion (and as special cases subject reduction and subject expansion). This follows immediately from subject conversion of HOL (which, as stated earlier, is itself a direct consequence of the [CONV] and [SUBST] rules).

Corollary 8 (*Subject conversion*)
Assume that $t_1 =_{\beta\iota\mu} t_1'$ and $t_2 =_{\beta\iota\mu} t_2'$ and $\Gamma \mid \Psi \vdash t_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi$. Then we have $\Gamma \mid \Psi \vdash t_1' : \sigma_1 \sim t_2' : \sigma_2 \mid \phi$.

Another useful consequence of the equivalence is the admissibility of the transitivity rule.

Corollary 9 (*Admissibility of transitivity rule*)
Assume that:

- $\Gamma \mid \Psi \vdash t_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi$
- $\Gamma \mid \Psi \vdash t_2 : \sigma_2 \sim t_3 : \sigma_3 \mid \phi'$

Then, $\Gamma \mid \Psi \vdash t_1 : \sigma_1 \sim t_3 : \sigma_3 \mid \phi[t_2/\mathbf{r}_2] \wedge \phi'[t_2/\mathbf{r}_1]$.

Finally, we prove an embedding lemma for UHOL. The proof can be carried by induction on the structure of derivations, or using the equivalence between UHOL and HOL (Theorem 3).

Lemma 10 (*Embedding lemma*)
Assume that:

- $\Gamma \mid \Psi \vdash t_1 : \sigma_1 \mid \phi$
- $\Gamma \mid \Psi \vdash t_2 : \sigma_2 \mid \phi'$

Then $\Gamma \mid \Psi \vdash t_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi[\mathbf{r}_1/\mathbf{r}] \wedge \phi'[\mathbf{r}_2/\mathbf{r}]$.

Proof
By the embedding into HOL (Theorem 3), we have:

- $\Gamma \mid \Psi \vdash \phi[t_1/\mathbf{r}]$
- $\Gamma \mid \Psi \vdash \phi'[t_2/\mathbf{r}]$

and by the $[\wedge_I]$ rule,

$$\Gamma \mid \Psi \vdash \phi[t_1/\mathbf{r}] \wedge \phi'[t_2/\mathbf{r}].$$

Finally, by Theorem 6:

$$\Gamma \mid \Psi \vdash t_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi[\mathbf{r}_1/\mathbf{r}] \wedge \phi'[\mathbf{r}_2/\mathbf{r}].$$

□

The embedding is reminiscent of the approach of (Beringer & Hofmann, 2007) to encode information flow properties in Hoare logic.

7 Embeddings

In this section, we establish the expressiveness of RHOL and UHOL by embedding several existing refinement type systems (three relational and one non-relational) from a variety of domains. All embeddings share the common idea of *separating* the simple typing information from the more fine-grained refinement information in the translation. We use uniform notation to represent similar ideas across the different embeddings. In particular, we use vertical bars $|\cdot|$ to denote the erasure of a type into a simple type, and floor bars $\lfloor\cdot\rfloor$ to denote the embedding of the refinement of a type in a HOL formula.

For clarity of exposition, we often present fragments or variants of systems that appear in the literature, notably excluding recursive functions that do not satisfy our well-definedness predicate. Moreover, the embeddings are given for a version of RHOL à la Curry, in which λ-abstractions do not carry the type of their bound variable.

7.1 Refinement Types

Refinement types (Freeman & Pfenning, 1991; Swamy *et al.*, 2016; Vazou *et al.*, 2014) are a variant of simple types where for every basic type τ, there is a type $\{x : \tau \mid \phi\}$ which is inhabited by the elements t of τ that satisfy the logical assertion $\phi[t/x]$. This includes dependent refinements $\Pi(x:\tau).\sigma$, in which the variable x is also bound in the refinement formulas appearing in σ. Here we present a simplified variant of these systems. (Refined) types are defined by the grammar

$$\tau := \mathbb{B} \mid \mathbb{N} \mid \mathsf{list}_\tau \mid \{x : \tau \mid \phi\} \mid \Pi(x:\tau).\tau$$

$$\frac{\Gamma \vdash \tau}{\Gamma \vdash \tau \preceq \tau} \qquad \frac{\Gamma \vdash \tau_1 \preceq \tau_2 \qquad \Gamma \vdash \tau_2 \preceq \tau_3}{\Gamma \vdash \tau_1 \preceq \tau_3} \qquad \frac{\Gamma \vdash \tau_1 \preceq \tau_2}{\Gamma \vdash \mathsf{list}_{\tau_1} \preceq \mathsf{list}_{\tau_2}}$$

$$\frac{\Gamma \vdash \{x : \tau \mid \phi\}}{\Gamma \vdash \{x : \tau \mid \phi\} \preceq \tau} \qquad \frac{\Gamma \vdash \tau \preceq \sigma \qquad \Gamma, x : \tau \vdash \phi}{\Gamma \vdash \tau \preceq \{x : \sigma \mid \phi\}} \qquad \frac{\Gamma \vdash \sigma_2 \preceq \sigma_1 \qquad \Gamma, x : \sigma_2 \vdash \tau_1 \preceq \tau_2}{\Gamma \vdash \Pi(x : \sigma_1).\tau_1 \preceq \Pi(x : \sigma_2).\tau_2}$$

$$\frac{}{\Gamma, x : \tau \vdash x : \tau} \qquad \frac{\Gamma, x : \tau \vdash t : \sigma}{\Gamma \vdash \lambda x.t : \Pi(x : \tau).\sigma} \qquad \frac{\Gamma \vdash t_1 : \Pi(x : \tau).\sigma \qquad \Gamma \vdash t_2 : \tau}{\Gamma \vdash t_1\ t_2 : \sigma[t_2/x]}$$

$$\frac{\Gamma \vdash t : \mathsf{list}_\tau \qquad \Gamma \vdash t_1 : \sigma \qquad \Gamma \vdash t_2 : \Pi(h : \tau).\Pi(l : \mathsf{list}_\tau).\sigma}{\Gamma \vdash \mathrm{case}\ t\ \mathrm{of}\ [] \mapsto t_1; \_ :: \_ \mapsto t_2 : \sigma} \qquad \frac{\Gamma \vdash \tau \preceq \sigma \qquad \Gamma \vdash t : \tau}{\Gamma \vdash t : \sigma}$$

$$\frac{\Gamma, x : \tau, f : \Pi(\{y : \tau \mid y < x\}).\sigma[y/x] \vdash t : \sigma \qquad \mathcal{D}ef(f, x, t)}{\Gamma \vdash \mathrm{letrec}\ f\ x = t : \Pi(x : \tau).\sigma}$$

Fig. 8. Refinement types rules (subtyping and typing)

As usual, we shorten $\Pi(x : \tau).\sigma$ to $\tau \to \sigma$ if $x \notin FV(\sigma)$. We also shorten bindings of the form $x : \{x : \tau \mid \phi\}$ to $\{x : \tau \mid \phi\}$. Typing rules are presented in Figure 8; note that the [LETREC] rule requires that recursive definitions satisfy the well-definedness predicate. Judgments of the form $\Gamma \vdash \tau$ are well-formedness judgments. Judgments of the form $\Gamma \vdash \phi$ are logical judgments; we omit a formal description of the rules, but assume that the logic of assertions is consistent with HOL, i.e. $\Gamma \vdash \phi$ implies $|\Gamma| \mid \lfloor \Gamma \rfloor \vdash \phi$, where the erasure functions are defined below.

This system can be embedded in UHOL. The embedding highlights the relation between these two systems, i.e. between logical assertions embedded in the types (as in refinement types) and logical assertions at the top-level, separate from simple types (as in UHOL). The intuitive idea behind the embedding is therefore to separate refinement assertions from types. Specifically, from every refinement type we can obtain a simple type by repeatedly extracting the type τ from $\{x : \tau \mid \phi\}$. We represent this extraction using the following translation function $|\tau|$:

$$|\mathbb{B}| \triangleq \mathbb{B} \qquad |\mathbb{N}| \triangleq \mathbb{N} \qquad |\mathsf{list}_\tau| \triangleq \mathsf{list}_{|\tau|} \qquad |\{x : \tau \mid \phi\}| \triangleq |\tau| \qquad |\Pi(x : \tau).\sigma| \triangleq |\tau| \to |\sigma|$$

Since $|\tau|$ loses refinement information, we define a second translation that extracts the refinement as a logical predicate over a variable x that names the typed expression. This second translation is written $\lfloor \tau \rfloor(x)$.

$$\lfloor \mathbb{B} \rfloor(x), \lfloor \mathbb{N} \rfloor(x) \triangleq \top \qquad \lfloor \mathsf{list}_\tau \rfloor(x) \triangleq \mathrm{All}(x, \lambda y.\lfloor \tau \rfloor(y))$$

$$\lfloor \{y : \tau \mid \phi\} \rfloor(x) \triangleq \lfloor \tau \rfloor(x) \wedge \phi[x/y] \qquad \lfloor \Pi(x : \tau).\sigma \rfloor(x) \triangleq \forall y.\lfloor \tau \rfloor(y) \Rightarrow \lfloor \sigma \rfloor(x\ y)$$

The refinement of simple types is trivial. If t is an expression of type $\{x : \tau \mid \phi\}$, then t must satisfy both the refinement formula ϕ and the refinement of τ. The refinement of a list uses the predicate All, which as defined in §4, means that all elements of a list satisfy a given formula. Finally, if t is an expression of type $\Pi(x : \tau).\sigma$, then it must be the case that for every x satisfying the refinement of τ, $(t\ x)$ satisfies the refinement of σ.

The syntax of assertions and expressions is exactly the same as in HOL, and therefore there is no need for a translation. The embedding of types can be lifted to contexts in the natural way.

$$|x:\tau,\Gamma| \triangleq x:|\tau|,|\Gamma| \qquad\qquad \lfloor x:\tau,\Gamma\rfloor \triangleq \lfloor\tau\rfloor(x),\lfloor\Gamma\rfloor$$

To encode judgments, all that remains is to put the previous definitions together. The main result about embedding typing judgments is the following:

Theorem 11
If $\Gamma \vdash t:\tau$ is derivable in the refinement type system, then $|\Gamma| \mid \lfloor\Gamma\rfloor \vdash t:|\tau| \mid \lfloor\tau\rfloor(\mathbf{r})$ is derivable in UHOL.

Proof
By induction on the derivation. We show the most interesting cases:

Case. $\dfrac{\Gamma,x:\tau\vdash t:\sigma}{\Gamma\vdash \lambda x.t:\Pi(x:\tau).\sigma}$
To prove: $|\Gamma| \mid \lfloor\Gamma\rfloor \vdash \lambda x.t : |\Pi(x:\tau).\sigma| \mid \lfloor\Pi(x:\tau).\sigma\rfloor(\mathbf{r})$.
Expanding the definitions:
$|\Gamma| \mid \lfloor\Gamma\rfloor \vdash \lambda x.t : |\tau|\to|\sigma| \mid \forall x.\lfloor\tau\rfloor(x)\Rightarrow\lfloor\sigma\rfloor(\mathbf{r}x)$
By induction hypothesis on the premise:
$|\Gamma|,x:|\tau| \mid \lfloor\Gamma\rfloor,\lfloor\tau\rfloor(x)\vdash t:|\sigma| \mid \lfloor\sigma\rfloor(\mathbf{r})$
Directly by [ABS].

Case. $\dfrac{\Gamma\vdash t:\Pi(x:\tau).\sigma \qquad \Gamma\vdash u:\tau}{\Gamma\vdash t\ u:\sigma[u/x]}$
To prove: $|\Gamma| \mid \lfloor\Gamma\rfloor \vdash t\ u : |\sigma[u/x]| \mid \lfloor\sigma[u/x]\rfloor(\mathbf{r})$.
Expanding the definitions:
$|\Gamma| \mid \lfloor\Gamma\rfloor \vdash t\ e_2 : |\sigma| \mid \lfloor\sigma\rfloor(\mathbf{r})[u/x]$
By induction hypothesis on the premise:
$|\Gamma| \mid \lfloor\Gamma\rfloor \vdash t : |\tau|\to|\sigma| \mid \forall x.\lfloor\tau\rfloor(x)\Rightarrow\lfloor\sigma\rfloor(\mathbf{r}x)$
and
$|\Gamma| \mid \lfloor\Gamma\rfloor \vdash u : |\tau| \mid \lfloor\tau\rfloor(\mathbf{r})$
We get the result directly by [APP].

Case. $\dfrac{\Gamma\vdash\tau\preceq\sigma \qquad \Gamma\vdash t:\tau}{\Gamma\vdash t:\sigma}$
To prove: $|\Gamma| \mid \lfloor\Gamma\rfloor \vdash t:|\sigma| \mid \lfloor\sigma\rfloor(\mathbf{r})$.
This follows from applying the induction hypothesis to the second premise, and then applying the rule [SUB] with Theorem 12 below. Note that the first premise implies $|\sigma|\equiv|\tau|$.

Case. $\dfrac{\Gamma,x:\tau,f:\Pi(y:\{\mathbf{r}:\tau\mid y<x\}).\sigma[y/x]\vdash t:\sigma \qquad \mathcal{D}ef(f,x,t)}{\Gamma\vdash \text{letrec}\ f\ x=t:\Pi(x:\tau).\sigma}$
To prove: $|\Gamma| \mid \lfloor\Gamma\rfloor \vdash \text{letrec}\ f\ x=t : |\Pi(x:\tau).\sigma| \mid \lfloor\Pi(x:\tau).\sigma\rfloor(\mathbf{r})$

By induction hypothesis on the premise:
$|\Gamma|, x : |\tau|, f : |\tau| \to |\sigma| \mid \lfloor\Gamma\rfloor, \lfloor\tau\rfloor(x), \forall y. \lfloor\tau\rfloor(y) \wedge y < x \Rightarrow \lfloor\sigma[y/x]\rfloor(fy) \vdash t : |\sigma| \mid \lfloor\sigma\rfloor(\mathbf{r})$
Directly by [LETREC]. □

The above proof relies on the following theorem about subtyping.

Theorem 12
If $\Gamma \vdash \tau \preceq \sigma$ is derivable in a refinement type system, then $|\Gamma|, x : |\tau| \mid \lfloor\Gamma\rfloor, \lfloor\tau\rfloor(x) \vdash \lfloor\sigma\rfloor(x)$ is derivable in HOL.

Proof
The proof is by induction on the derivation. We show here only the case of the dependent product:

Case. $\dfrac{\Gamma \vdash \sigma_2 \preceq \sigma_1 \qquad \Gamma, x : \sigma_2 \vdash \tau_1 \preceq \tau_2}{\Gamma \vdash \Pi(x : \sigma_1).\tau_1 \preceq \Pi(x : \sigma_2).\tau_2}$
To show: $|\Gamma|, f : |\Pi(x : \sigma_1).\tau_1| \mid \lfloor\Gamma\rfloor, \lfloor\Pi(x : \sigma_1).\tau_1\rfloor(f) \vdash \lfloor\Pi(x : \sigma_2).\tau_2\rfloor(f)$
Expanding the definitions:
$|\Gamma|, f : |\Pi(x : \sigma_1).\tau_1| \mid \lfloor\Gamma\rfloor, \forall x. \lfloor\sigma_1\rfloor(x) \Rightarrow \lfloor\tau_1\rfloor(fx) \vdash \forall x. \lfloor\sigma_2\rfloor(x) \Rightarrow \lfloor\tau_2\rfloor(fx)$
By the rules $[\forall_I]$ and $[\Rightarrow_I]$ it suffices to prove:
$|\Gamma|, f : |\Pi(x : \sigma_1).\tau_1|, x : |\sigma_2| \mid \lfloor\Gamma\rfloor, \forall x. \lfloor\sigma_1\rfloor(x) \Rightarrow \lfloor\tau_1\rfloor(fx), \lfloor\sigma_2\rfloor(x) \vdash \lfloor\tau_2\rfloor(fx)$ (1)
On the other hand, by induction hypothesis on the premises:
$|\Gamma|, x : |\sigma_2| \mid \lfloor\Gamma\rfloor, \lfloor\sigma_2\rfloor(x) \vdash \lfloor\sigma_1\rfloor(x)$ (2)
and
$|\Gamma|, x : |\sigma_2|, y : |\tau_1| \mid \lfloor\Gamma\rfloor, \lfloor\sigma_2\rfloor(x), \lfloor\tau_1\rfloor(y) \vdash \lfloor\tau_2\rfloor(y)$ (3)
which we can weaken respectively to:
$|\Gamma|, x : |\sigma_2|, f : |\Pi(x : \sigma_1).\tau_1| \mid |\Gamma|, \lfloor\sigma_2\rfloor(x), \forall x. \lfloor\sigma_1\rfloor(x) \Rightarrow \lfloor\tau_1\rfloor(fx) \vdash \lfloor\sigma_1\rfloor(x)$ (4)
and
$|\Gamma|, x : |\sigma_2|, y : |\tau_1|, f : |\Pi(x : \sigma_1).\tau_1| \mid |\Gamma|, \lfloor\sigma_2\rfloor(x), \lfloor\tau_1\rfloor(y), \forall x. \lfloor\sigma_1\rfloor(x) \Rightarrow \lfloor\tau_1\rfloor(fx) \vdash$
$\lfloor\tau_2\rfloor(y)$ (5)
From (4), by doing a cut with its own premise $\forall x. \lfloor\sigma_1\rfloor(x) \Rightarrow \lfloor\tau_1\rfloor(fx)$, we derive:
$|\Gamma|, x : |\sigma_2|, f : |\Pi(x : \sigma_1).\tau_1| \mid \lfloor\Gamma\rfloor, \lfloor\sigma_2\rfloor(x), \forall x. \lfloor\sigma_1\rfloor(x) \Rightarrow \lfloor\tau_1\rfloor(fx) \vdash \lfloor\tau_1\rfloor(fx)$ (6)
From (5), by $[\Rightarrow_I]$ and $[\forall_I]$ we can derive:
$|\Gamma|, x : |\sigma_2|, f : |\Pi(x : \sigma_1).\tau_1| \mid \lfloor\Gamma\rfloor, \lfloor\sigma_2\rfloor(x),, \forall x. \lfloor\sigma_1\rfloor(x) \Rightarrow \lfloor\tau_1\rfloor(fx) \vdash \forall y. \lfloor\tau_1\rfloor(y) \Rightarrow$
$\lfloor\tau_2\rfloor(y)$
And by $[\forall_E]$
$|\Gamma|, x : |\sigma_2|, f : |\Pi(x : \sigma_1).\tau_1| \mid \lfloor\Gamma\rfloor, \lfloor\sigma_2\rfloor(x),, \forall x. \lfloor\sigma_1\rfloor(x) \Rightarrow \lfloor\tau_1\rfloor(fx) \vdash \lfloor\tau_1\rfloor(fx) \Rightarrow \lfloor\tau_2\rfloor(fx)$
(7)
Finally, from (6) and (7) by $[\Rightarrow_E]$ we get:
$|\Gamma|, x : |\sigma_2|, f : |\Pi(x : \sigma_1).\tau_1| \mid \lfloor\Gamma\rfloor, \lfloor\sigma_2\rfloor(x), \forall x. \lfloor\sigma_1\rfloor(x) \Rightarrow \lfloor\tau_1\rfloor(fx) \vdash \lfloor\tau_2\rfloor(fx)$
and by one last application of $[\Rightarrow_I]$ we get what we wanted to prove. □

Soundness of refinement types w.r.t. the set-theoretical semantics follows immediately from Theorem 11 and the set-theoretical soundness of UHOL (Corollary 4).

Corollary 13 (*Soundness of refinement types*)
If $\Gamma \vdash t : \tau$, then for every valuation $\rho \models \Gamma$ we have $[\![t]\!]_\rho \in [\![\tau]\!]_\rho$.

As a final remark, note that a function with a refinement type can be interpreted in two different ways: 1) As a map whose domain is the domain type restricted to its (the type's) refinement, or 2) As a map whose domain is the entire domain type (disregarding the refinement), but whose result satisfies the co-domain's refinement only if the argument satisfies the domain's refinement. We use the second interpretation, while some prior work, for example (Freeman & Pfenning, 1991), uses the first. Type systems using the first interpretation can inhabit types that this embedding would erase into uninhabited simple types. For instance, consider the type $\{x : \mathbb{N} \mid x > 1 \land x < 1\} \to \mathsf{Emp}$ where Emp denotes the empty type. This is inhabited (under appropriate subtyping rules), but it would be erased into the type $\mathbb{N} \to \mathsf{Emp}$, which is not inhabited.

7.2 Relational Refinement Types

Relational refinement types (Barthe *et al.*, 2014; Barthe *et al.*, 2015) are a variant of refinement types that can be used to express relational properties via a syntax of the form $\{\mathbf{r} :: \tau \mid \phi\}$ where ϕ is a relational assertion—i.e. it may contain a left and right copy of $\mathbf{r}$, which are denoted as $\mathbf{r}_1$ and $\mathbf{r}_2$ respectively, as well as a left and a right copy of every variable in the context. In this section, we introduce a simple relational refinement type system and establish a type-preserving translation to RHOL.

The syntax of relational refinement types is given by the grammar:

$$\begin{aligned} \tau &::= \mathbb{B} \mid \mathbb{N} \mid \tau \to \tau \\ T, U &::= \tau \mid \mathsf{list}_T \mid \Pi(x :: T).\, U \mid \{x :: T \mid \phi\} \end{aligned}$$

Relational refinement types are naturally ordered by a subtyping relation $\Gamma \vdash T \preceq U$, where Γ is a sequence of variables declarations of the form $x :: U$.

Typing judgments are of the form $\Gamma \vdash t_1 \sim t_2 :: T$. We present selected typing rules in Figure 9. Note that the form of judgments requires that t_1 and t_2 have the same simple type, and the typing rules require that t_1 and t_2 have the same structure[1]. In the [$\mathsf{CASELIST}$] rule, we require that both terms reduce to the same branch; the case rule for natural numbers is similar. The [LETREC] rule uses (a straightforward adaptation of) the $\mathcal{D}ef(f, x, t)$ predicate from our simply-typed language, and requires that the two recursive definitions perform exactly the same recursive calls.

Subtyping rules are the same as in the unary case, and therefore we refer to Figure 8 for them (allowing their instantiation for relational types T, U as well as unary types σ, τ).

[1] The typing rules displayed in the figure will in fact force t_1 and t_2 to be the same term modulo renaming. This is not the case in existing relational refinement type systems; however, rules that introduce different terms on the right and on the left are limited, since both terms still need to have the same type and most one-sided rules break this invariant. For instance, in (Barthe *et al.*, 2014) there is a rule similar to [LLCASE-A], and a rule for reducing one of the terms of a judgment.

$$\text{VAR-RT}\ \frac{(x:T)\in\Gamma}{\Gamma\vdash x_1\sim x_2::T} \qquad \text{ABS-RT}\ \frac{\Gamma,x::T\vdash u_1\sim u_2::U}{\Gamma\vdash \lambda x_1.u_1\sim\lambda x_2.u_2::\Pi(x::T).U}$$

$$\text{APP-RT}\ \frac{\Gamma\vdash t_1\sim t_2::\Pi(x::T).U \qquad \Gamma\vdash u_1\sim u_2::T}{\Gamma\vdash t_1\ u_1\sim t_2\ u_2::U[u_1/x_1][u_2/x_2]} \qquad \text{NIL}\ \frac{\Gamma\vdash T}{\Gamma\vdash []\sim[]::\mathsf{list}_T}$$

$$\text{CONS}\ \frac{\Gamma\vdash h_1\sim h_2::T \qquad \Gamma\vdash t_1\sim t_2::\mathsf{list}_T}{\Gamma\vdash h_1::t_1\sim h_2::t_2::\mathsf{list}_T} \qquad \textsc{Sub}\ \frac{\Gamma\vdash t_1\sim t_2::T \qquad \Gamma\vdash T\preceq U}{\Gamma\vdash t_1\sim t_2::U}$$

$$\text{LETREC-RT}\ \frac{\begin{array}{c}\Gamma,x::T,f::\Pi(\{y::T\mid(y_1,y_2)<(x_1,x_2)\}).U[y/x]\vdash t_1\sim t_2::U\\ \Gamma\vdash\Pi(x::T).U \qquad \mathcal{D}ef(f,x,t)\end{array}}{\Gamma\vdash \text{letrec } f_1\ x_1=t_1\sim\text{letrec } f_2\ x_2=t_2::\Pi(x::T).U}$$

$$\text{CASELIST}\ \frac{\begin{array}{c}\Gamma\vdash t_1\sim t_2::\mathsf{list}_\tau \qquad \Gamma\vdash t_1=[]\Leftrightarrow t_2=[]\\ \Gamma\vdash u_1\sim u_2::T \qquad \Gamma\vdash v_1\sim v_2::\Pi(h::\tau).\Pi(t::\mathsf{list}_\tau).T\end{array}}{\Gamma\vdash \text{case } t_1 \text{ of } []\mapsto u_1;\_::\_\mapsto v_1\sim\text{case } t_2 \text{ of } []\mapsto u_2;\_::\_\mapsto v_2::T}$$

Fig. 9. Relational Typing (Selected Rules)

The embedding of refinement types into UHOL can be adapted to the relational setting. From each relational refinement type T we can extract a simple type $|T|$. On the other hand, we can erase every relational refinement type T into a relational formula $\llbracket T\rrbracket$, which is parametrized by two expressions and defined as follows:

$$\llbracket\mathsf{list}_\tau\rrbracket(x_1,x_2)\triangleq\bigwedge_{i\in\{1,2\}}\mathrm{All}(x_i,\lambda y.\lfloor\tau\rfloor(y))$$

$$\llbracket\mathsf{list}_T\rrbracket(x_1,x_2)\triangleq\mathrm{All2}(x_1,x_2,\lambda y_1.\lambda y_2.\llbracket T\rrbracket(y_1,y_2))$$

$$\llbracket\{y:\tau\mid\phi\}\rrbracket(x_1,x_2)\triangleq\bigwedge_{i\in\{1,2\}}\lfloor\tau\rfloor(x_i)\wedge\phi[x_i/y]$$

$$\llbracket\{y::T\mid\phi\}\rrbracket(x_1,x_2)\triangleq\llbracket T\rrbracket(x_1,x_2)\wedge\phi[x_1/y_1][x_2/y_2]$$

$$\llbracket\Pi(y:\tau).\sigma\rrbracket(x)\triangleq\bigwedge_{i\in\{1,2\}}\forall y.\lfloor\tau\rfloor(y)\Rightarrow\lfloor\sigma\rfloor(xy)$$

$$\llbracket\Pi(y::T).U\rrbracket(x_1,x_2)\triangleq\forall y_1y_2.\llbracket T\rrbracket(y_1,y_2)\Rightarrow\llbracket U\rrbracket(x_1y_1,\ x_2y_2)$$

The predicate All2 relates two lists element-wise and is defined axiomatically:

$$\mathrm{All2}([],[],\lambda x_1.\lambda x_2.\phi)$$

$$\forall h_1h_2t_1t_2.\,\mathrm{All2}(t_1,t_2,\lambda x_1.\lambda x_2.\phi)\Rightarrow\phi(h_1,h_2)\Rightarrow\mathrm{All2}(h_1::t_1,h_2::t_2,\lambda x_1.\lambda x_2.\phi)$$

To extend the embedding to contexts, we need to duplicate every variable in them:

$$|x :: T, \Gamma| \triangleq x_1, x_2 : |T|, |\Gamma| \qquad\qquad \llbracket x :: T, \Gamma \rrbracket \triangleq \llbracket T \rrbracket(x_1, x_2), \llbracket \Gamma \rrbracket$$

Now we state the main result:

Theorem 14 (*Soundness of embedding*)
If $\Gamma \vdash t_1 \sim t_2 :: T$, then $|\Gamma| \mid \llbracket \Gamma \rrbracket \vdash t_1 : |T| \sim t_2 : |T| \mid \llbracket T \rrbracket(\mathbf{r}_1, \mathbf{r}_2)$. Also, if $\Gamma \vdash T \preceq U$ then $|\Gamma|, x_1, x_2 : |T| \mid \llbracket \Gamma \rrbracket, \llbracket T \rrbracket(x_1, x_2) \vdash \llbracket U \rrbracket(x_1.x_2)$.

Proof
The proof proceeds by induction on the structure of derivations. Most cases are very similar to the unary case, so we will only show the most interesting ones:

Case. $\dfrac{\Gamma \vdash T}{\Gamma \vdash [] \sim [] :: \mathsf{list}_T}$
To show: $|\Gamma| \mid \llbracket \Gamma \rrbracket \vdash [] : |\mathsf{list}_T| \sim [] : |\mathsf{list}_T| \mid \llbracket \mathsf{list}_T \rrbracket(\mathbf{r}_1, \mathbf{r}_2)$.
There are two options. If T is a unary type, we have to prove:
$|\Gamma| \mid \llbracket \Gamma \rrbracket \vdash [] : |\mathsf{list}_T| \sim [] : |\mathsf{list}_T| \mid \bigwedge_{i \in \{1,2\}} \mathrm{All}(\mathbf{r}_i, \lambda x. \lfloor \tau \rfloor(x))$
And by the definition of All we can directly prove:
$\emptyset \mid \emptyset \vdash \mathrm{All}([], \lambda x. \lfloor \tau \rfloor(x)) \wedge \mathrm{All}([], \lambda x. \lfloor \tau \rfloor(x))$
If T is a relational type, we have to prove:
$|\Gamma| \mid \llbracket \Gamma \rrbracket \vdash [] : |\mathsf{list}_T| \sim [] : |\mathsf{list}_T| \mid \mathrm{All2}(\mathbf{r}_1, \mathbf{r}_2, \lambda x_1. \lambda x_2. \llbracket T \rrbracket(x_1, x_2))$
And by the definition of All2 we can directly prove:
$\emptyset \mid \emptyset \vdash \mathrm{All2}([], [], \lambda x_1. \lambda x_2. \llbracket T \rrbracket(x_1, x_2))$

Case. $\dfrac{\Gamma \vdash h_1 \sim h_2 :: T \qquad \Gamma \vdash t_1 \sim t_2 :: \mathsf{list}_T}{\Gamma \vdash h_1 :: t_1 \sim h_2 :: t_2 :: \mathsf{list}_T}$
To show: $|\Gamma| \mid \llbracket \Gamma \rrbracket \vdash h_1 :: t_2 : |\mathsf{list}_T| \sim h_2 :: t_2 : |\mathsf{list}_T| \mid \mathsf{list}_T$.
There are two options. If T is a unary type, we have to prove:
$|\Gamma| \mid \llbracket \Gamma \rrbracket \vdash h_1 :: t_1 : |\mathsf{list}_T| \sim h_2 :: t_2 : |\mathsf{list}_T| \mid \bigwedge_{i \in \{1,2\}} \mathrm{All}(\mathbf{r}_i, \lambda x. \lfloor T \rfloor(x))$
By induction hypothesis we have:
$|\Gamma| \mid \llbracket \Gamma \rrbracket \vdash h_1 : |T| \sim h_2 :: t_2 : |T| \mid \bigwedge_{i \in \{1,2\}} \lfloor T \rfloor(\mathbf{r}_i)$
and
$|\Gamma| \mid \llbracket \Gamma \rrbracket \vdash t_1 : |\mathsf{list}_T| \sim t_2 : |\mathsf{list}_T| \mid \bigwedge_{i \in \{1,2\}} \mathrm{All}(\mathbf{r}_i, \lambda x. \lfloor T \rfloor(x))$
And by the definition of All we can directly prove:
$\bigwedge_{i \in \{1,2\}} \lfloor T \rfloor(h_i) \Rightarrow \bigwedge_{i \in \{1,2\}} \mathrm{All}(t_i, \lambda x. \lfloor T \rfloor(x)) \Rightarrow \bigwedge_{i \in \{1,2\}} \mathrm{All}(h_i :: t_i, \lambda x. \lfloor T \rfloor(x))$
So by the [CONS] rule, we prove the result. If T is a relational type, we have to prove:
$|\Gamma| \mid \llbracket \Gamma \rrbracket \vdash h_1 :: t_1 : |\mathsf{list}_T| \sim h_2 :: t_2 : |\mathsf{list}_T| \mid \mathrm{All2}(\mathbf{r}_1, \mathbf{r}_2, \lambda x_1. \lambda x_2. \llbracket T \rrbracket(x_1, x_2))$
By induction hypothesis we have:
$|\Gamma| \mid \llbracket \Gamma \rrbracket \vdash h_1 : |T| \sim h_2 :: t_2 : |T| \mid \llbracket T \rrbracket(\mathbf{r}_1, \mathbf{r}_2)$
and
$|\Gamma| \mid \llbracket \Gamma \rrbracket \vdash t_1 : |\mathsf{list}_T| \sim t_2 : |\mathsf{list}_T| \mid \mathrm{All2}(\mathbf{r}_1, \mathbf{r}_2, \lambda x_1. \lambda x_2. \llbracket T \rrbracket(x_1, x_2))$
And by the definition of All2 we can directly prove:
$\llbracket T \rrbracket(h_1, h_2) \Rightarrow \mathrm{All2}(t_1, t_2, \lambda x_1. \lambda x_2. \llbracket T \rrbracket(x_1, x_2)) \Rightarrow \mathrm{All}(h_1 :: t_1, h_1 :: h_2, \lambda x_1. \lambda x_2. \llbracket T \rrbracket(x_1, x_2))$

So by the [CONS] rule, we prove the result.

Case. $$\frac{\Gamma, x :: T, f :: \Pi(y :: \{y :: T \mid (y_1, y_2) < (x_1, x_2)\}).U[y/x] \vdash t_1 \sim t_2 :: U \quad \Gamma \vdash \Pi(x :: T).U \quad \mathcal{D}ef(f_1, x_1, t_1) \quad \mathcal{D}ef(f_2, x_2, t_2)}{\Gamma \vdash \text{letrec } f_1\ x_1 = t_1 \sim \text{letrec } f_2\ x_2 = t_2 :: \Pi(x :: T).U}$$

To show:
$|\Gamma| \mid \llbracket \Gamma \rrbracket \vdash \text{letrec } f_1\ x_1 = t_1 : |\Pi(x :: T).U| \sim \text{letrec } f_2\ x_2 = t_2 : |\Pi(x :: T).U| \mid \llbracket \Pi(x :: T).U \rrbracket(\mathbf{r}_1, \mathbf{r}_2)$
Expanding the definitions:
$|\Gamma| \mid \llbracket \Gamma \rrbracket \vdash \text{letrec } f_1\ x_1 = t_1 : |T| \to |U| \sim \text{letrec } f_2\ x_2 = t_2 : |T| \to |U| \mid \forall x_1 x_2. \llbracket T \rrbracket(x_1, x_2) \Rightarrow \llbracket U \rrbracket(\mathbf{r}_1 x_1,\ \mathbf{r}_2 x_2)$
By induction hypothesis on the premise:
$|\Gamma|, x_1, x_2 : |T|, f_1, f_2 : |T| \to |U| \mid \llbracket \Gamma \rrbracket, \llbracket T \rrbracket(x_1, x_2), \forall y_1, y_2. (\llbracket T \rrbracket(y_1, y_2) \wedge (y_1, y_2) < (x_1, x_2)) \Rightarrow \llbracket U \rrbracket(f_1 x_1,\ f_2 x_2) \vdash t_1 : |U| \sim t_2 : |U| \mid \llbracket U \rrbracket(\mathbf{r}_1, \mathbf{r}_2)$
And we apply the [LETREC] rule to get the result. $\square$

Soundness of relational refinement types w.r.t. set-theoretical semantics follows immediately from Theorem 14 and the set-theoretical soundness of RHOL (Corollary 7).

Corollary 15 (*Soundness of relational refinement types*)
If $\Gamma \vdash t_1 \sim t_2 :: T$, then for every valuation $\theta \models \Gamma$ we have $((\!|t_1|\!)_\theta, (\!|t_2|\!)_\theta) \in (\!|T|\!)_\theta$.

7.3 Dependency Core Calculus

The Dependency Core Calculus (DCC) (Abadi *et al.*, 1999) is a higher-order calculus with a type system that tracks data dependencies. DCC was designed as a unifying framework for dependency analysis and it was shown that many other calculi for information flow analysis (Heintze & Riecke, 1998; Volpano *et al.*, 1996), binding-time analysis (Hatcliff & Danvy, 1997), and program slicing, all of which track dependencies, can be embedded in DCC. Here, we show how a fragment of DCC can be embedded into RHOL. Transitively, the corresponding fragments of all the aforementioned calculi can also be embedded in RHOL. The fragment of DCC we consider excludes recursive functions. DCC admits general recursive functions, while our definition of RHOL only admits a subset of these. Extending the embedding to recursive functions admitted by RHOL is not difficult.

DCC is an extension of the simply typed lambda-calculus with a monadic type family $\mathbb{T}_\ell(\tau)$, indexed by *labels* ℓ, which are elements of a lattice. Unlike other uses of monads, DCC's monad does not encapsulate any effects. Instead, its only goal is to track dependence. The type system forces that the result of an expression of type $\mathbb{T}_\ell(\tau)$ can *depend* on an expression of type $\mathbb{T}_{\ell'}(\tau')$ only if $\ell' \sqsubseteq \ell$ in the lattice. Dually, if $\ell' \not\sqsubseteq \ell$, then even if an expression e of type $\mathbb{T}_\ell(\tau)$ mentions a variable x of type $\mathbb{T}_{\ell'}(\tau')$, then e's *result* must be independent of the substitution provided for x during evaluation.

For simplicity and without any loss of generality, we consider here only a two point lattice $\{L, H\}$ with $L \sqsubset H$. The syntax of DCC's types and expressions is shown below. We use e to denote DCC expressions, to avoid confusion with HOL's expressions.

$$\begin{array}{lll} \tau & ::= & \mathbb{B} \mid \tau \to \tau \mid \tau \times \tau \mid \mathbb{T}_\ell(\tau) \\ e & ::= & x \mid \lambda x.e \mid e_1\ e_2 \mid \mathsf{tt} \mid \mathsf{ff} \mid \mathsf{case}\ e\ \mathsf{of}\ \mathsf{tt} \mapsto e_t; \mathsf{ff} \mapsto e_f \mid \langle e_1, e_2 \rangle \mid \pi_1(e) \mid \pi_2(e) \\ & & \mid \eta_\ell(e) \mid \mathsf{bind}(e_1, x.e_2) \end{array}$$

Here, $\eta_\ell(e)$ and $\mathsf{bind}(e_1, x.e_2)$ are respectively the return and bind constructs for the monad $\mathbb{T}_\ell(\tau)$. Typing rules for these two constructs are shown below. Typing rules for the remaining constructs are the standard ones.

$$\frac{\Gamma \vdash e : \tau}{\Gamma \vdash \eta_\ell(e) : \mathbb{T}_\ell(\tau)} \qquad \frac{\Gamma \vdash e_1 : \mathbb{T}_\ell(\tau_1) \qquad \Gamma, x : \tau_1 \vdash e_2 : \tau_2 \qquad \tau_2 \searrow \ell}{\Gamma \vdash \mathsf{bind}(e_1, x.e_2) : \tau_2}$$

The crux of the dependency tracking is the relation $\tau_2 \searrow \ell$ in the premise of the rule for bind. The relation, read "τ_2 protected at level ℓ" and defined below, informally means that all primitive (boolean) values extractable from e_2 are protected by a monadic construct of the form $\mathbb{T}_{\ell'}(\tau)$, with $\ell \sqsubseteq \ell'$. Hence, the rule forces that the result obtained by eliminating the type $\mathbb{T}_\ell(\tau_1)$ flow only into types protected at ℓ in this sense.

$$\frac{\ell \sqsubseteq \ell'}{\mathbb{T}_{\ell'}(\tau) \searrow \ell} \qquad \frac{\tau \searrow \ell}{\mathbb{T}_{\ell'}(\tau) \searrow \ell} \qquad \frac{\tau_1 \searrow \ell \qquad \tau_2 \searrow \ell}{\tau_1 \times \tau_2 \searrow \ell} \qquad \frac{\tau_2 \searrow \ell}{\tau_1 \to \tau_2 \searrow \ell}$$

This fragment of DCC has a relational set-theoretical interpretation. For every type τ, we define a carrier set $|\tau|$:

$$|\mathbb{B}| \triangleq \mathbb{B} \qquad |\tau_1 \to \tau_2| \triangleq |\tau_1| \to |\tau_2| \qquad |\tau_1 \times \tau_2| \triangleq |\tau_1| \times |\tau_2| \qquad |\mathbb{T}_\ell(\tau)| \triangleq |\tau|$$

Next, every type τ is interpreted as a lattice-indexed family of relations $\lfloor \tau \rfloor_a \subseteq |\tau| \times |\tau|$. The role of the lattice element a is that it defines what can be *observed* in the system. Specifically, an expression of type $\mathbb{T}_\ell(\tau)$ can be observed only if $\ell \sqsubseteq a$. When $\ell \not\sqsubseteq a$, expressions of type $\mathbb{T}_\ell(\tau)$ look like "black-boxes". Technically, we force $\lfloor \mathbb{T}_\ell(\tau) \rfloor_a = |\tau| \times |\tau|$ when $\ell \not\sqsubseteq a$. DCC's typing rules are sound with respect to this model. The soundness implies that if $\ell \not\sqsubseteq \ell'$ and $x : \mathbb{T}_\ell(\mathbb{B}) \vdash e : \mathbb{T}_{\ell'}(\mathbb{B})$, then for $e_1, e_2 : \mathbb{T}_\ell(\mathbb{B})$, $e[e_1/x]$ and $e[e_2/x]$ are equal booleans in the set-theoretical model. This result, called noninterference, formalizes that DCC's dependency tracking is correct.

To translate DCC to RHOL, we actually embed this set-theoretical model in RHOL. We start by defining an erasing translation, $|\tau|$, from DCC's types into RHOL's simple types. This translation is exactly the same as the definition of carrier sets shown above, except that we treat $\times$ and $\to$ as RHOL's syntactic type constructs instead of set-theoretical constructs. Next, we define an erasure of terms:

$$|\mathsf{tt}| \triangleq \mathsf{tt} \qquad |\mathsf{ff}| \triangleq \mathsf{ff} \qquad |\text{case } e \text{ of } \mathsf{tt} \mapsto e_t; \mathsf{ff} \mapsto e_f| \triangleq \text{case } |e| \text{ of } \mathsf{tt} \mapsto |e_t|; \mathsf{ff} \mapsto |e_f|$$

$$|x| \triangleq x \qquad |\lambda x.e| \triangleq \lambda x.|e| \qquad |e_1\ e_2| \triangleq |e_1|\ |e_2| \qquad |\langle e_1, e_2\rangle| \triangleq \langle |e_1|, |e_2|\rangle$$

$$|\pi_1(e)| \triangleq \pi_1(|e|) \qquad |\pi_2(e)| \triangleq \pi_2(|e|) \qquad |\eta_\ell(e)| \triangleq |e|$$

$$|\mathsf{bind}(e_1, x.e_2)| \triangleq (\lambda x.|e_2|)\ |e_1|$$

It is fairly easy to see that if $\vdash e : \tau$ in DCC, then $\vdash |e| : |\tau|$. Next, we define the lattice-indexed family of relations $\lfloor \tau \rfloor_a$ in HOL. For technical convenience, we write the relations as logical assertions, indexed by variables x, y representing the two terms to be related.

$$\lfloor \mathbb{B} \rfloor_a(x,y) \triangleq (x = \mathsf{tt} \wedge y = \mathsf{tt}) \vee (x = \mathsf{ff} \wedge y = \mathsf{ff})$$

$$\lfloor \tau_1 \to \tau_2 \rfloor_a(x,y) \triangleq \forall v\, w.\, \lfloor \tau_1 \rfloor_a(v,w) \Rightarrow \lfloor \tau_2 \rfloor_a(x\ v, y\ w)$$

$$\lfloor \tau_1 \times \tau_2 \rfloor_a(x,y) \triangleq \lfloor \tau_1 \rfloor_a(\pi_1(x), \pi_1(y)) \wedge \lfloor \tau_2 \rfloor_a(\pi_2(x), \pi_2(y))$$

$$\lfloor \mathbb{T}_\ell(\tau) \rfloor_a(x,y) \triangleq \begin{cases} \lfloor \tau \rfloor_a(x,y) & \ell \sqsubseteq a \\ \top & \ell \not\sqsubseteq a \end{cases}$$

The most important clause is the last one: When $\ell \not\sqsubseteq a$, any two x, y are in the relation $\lfloor \mathbb{T}_\ell(\tau) \rfloor_a$. This generalizes to all protected types in the following sense.

Lemma 16
If $\ell \not\sqsubseteq a$ and $\tau \searrow \ell$, then $\vdash \forall x\, y.(\lfloor \tau \rfloor_a(x,y) \equiv \top)$ in HOL.

Proof
By induction on the derivation of $\tau \searrow \ell$.

Case. $\dfrac{\ell \sqsubseteq \ell'}{\mathbb{T}_{\ell'}(\tau) \searrow \ell}$

Since $\ell \not\sqsubseteq a$ (given) and $\ell \sqsubseteq \ell'$ (premise), it must be the case that $\ell' \not\sqsubseteq a$. Hence, by definition, $\lfloor \mathbb{T}_{\ell'}(\tau) \rfloor_a(x,y) = \top$.

Case. $\dfrac{\tau \searrow \ell}{\mathbb{T}_{\ell'}(\tau) \searrow \ell}$

We consider two cases:

If $\ell' \not\sqsubseteq a$, then $\lfloor \mathbb{T}_{\ell'}(\tau) \rfloor_a(x,y) = \top$ by definition.

If $\ell' \sqsubseteq a$, then $\lfloor \mathbb{T}_{\ell'}(\tau) \rfloor_a(x,y) = \lfloor \tau \rfloor_a(x,y)$ by definition. By i.h. on the premise, we have $\lfloor \tau \rfloor_a(x,y) \equiv \top$. Composing, $\lfloor \mathbb{T}_{\ell'}(\tau) \rfloor_a(x,y) \equiv \top$.

Case. $\dfrac{\tau_1 \searrow \ell \qquad \tau_2 \searrow \ell}{\tau_1 \times \tau_2 \searrow \ell}$

By i.h. on the premises, we have $\lfloor\tau_i\rfloor_a(x,y) \equiv \top$ for $i = 1,2$ and all x,y. Therefore, $\lfloor\tau_1 \times \tau_2\rfloor_a(x,y) \triangleq \lfloor\tau_1\rfloor_a(\pi_1(x),\pi_1(y)) \wedge \lfloor\tau_2\rfloor_a(\pi_2(x),\pi_2(y)) \equiv \top \wedge \top \equiv \top$.

Case. $\dfrac{\tau_2 \searrow \ell}{\tau_1 \to \tau_2 \searrow \ell}$

By i.h. on the premise, we have $\lfloor\tau_2\rfloor_a(x,y) \equiv \top$ for all x,y. Hence, $\lfloor\tau_1 \to \tau_2\rfloor_a(x,y) \triangleq (\forall v,w.\, \lfloor\tau_1\rfloor_a(v,w) \Rightarrow \lfloor\tau_2\rfloor_a(x\ v, y\ w)) \equiv (\forall v,w.\, \lfloor\tau_1\rfloor_a(v,w) \Rightarrow \top) \equiv \top$. □

The translations extend to contexts as follows:

$$|x^1:\tau_1,\ldots,x^n:\tau_n| \triangleq x^1_1:|\tau_1|, x^1_2:|\tau_1|,\ldots,x^n_1:|\tau_n|, x^n_2:|\tau_n|$$
$$\lfloor x^1:\tau_1,\ldots,x^n:\tau_n\rfloor_a \triangleq \lfloor\tau_1\rfloor_a(x^1_1,x^1_2),\ldots,\lfloor\tau_n\rfloor_a(x^n_1,x^n_2)$$

The following theorem states that the whole translation is sound: It preserves well-typedness. In the statement of the theorem, $|e|_1$ and $|e|_2$ replace each variable x in $|e|$ with x_1 and x_2, respectively.

Theorem 17 (*Soundness of embedding*)
If $\Gamma \vdash e:\tau$ in DCC, then for all $a \in \{L,H\}$: $|\Gamma| \mid \lfloor\Gamma\rfloor_a \vdash |e|_1 : |\tau| \sim |e|_2 : |\tau| \mid \lfloor\tau\rfloor_a(\mathbf{r}_1,\mathbf{r}_2)$.

Proof
By induction on the given derivation of $\Gamma \vdash e:\tau$. We show here the cases corresponding to the monadic constructions only:

Case. $\dfrac{\Gamma \vdash e:\tau}{\Gamma \vdash \eta_\ell(e) : \mathbb{T}_\ell(\tau)}$

To show: $|\Gamma| \mid \lfloor\Gamma\rfloor_a \vdash |e|_1 : |\tau| \sim |e|_2 : |\tau| \mid \lfloor\mathbb{T}_\ell(\tau)\rfloor_a(\mathbf{r}_1,\mathbf{r}_2)$.
By i.h. on the premise: $|\Gamma| \mid \lfloor\Gamma\rfloor_a \vdash |e|_1 : |\tau| \sim |e|_2 : |\tau| \mid \lfloor\tau\rfloor_a(\mathbf{r}_1,\mathbf{r}_2)$ (1)
If $\ell \sqsubseteq a$, then $\lfloor\mathbb{T}_\ell(\tau)\rfloor_a(\mathbf{r}_1,\mathbf{r}_2) \triangleq \lfloor\tau\rfloor_a(\mathbf{r}_1,\mathbf{r}_2)$, so the required result is the same as (1).
If $\ell \not\sqsubseteq a$, then $\lfloor\mathbb{T}_\ell(\tau)\rfloor_a(\mathbf{r}_1,\mathbf{r}_2) \triangleq \top$ and the required result follows from rule SUB applied to (1).

Case. $\dfrac{\Gamma \vdash e:\mathbb{T}_\ell(\tau) \qquad \Gamma, x:\tau \vdash e':\tau' \qquad \tau' \searrow \ell}{\Gamma \vdash \mathsf{bind}(e, x.e') : \tau'}$

To show: $|\Gamma| \mid \lfloor\Gamma\rfloor_a \vdash (\lambda x.|e'|_1)\ |e|_1 : |\tau'| \sim (\lambda x.|e'|_2)\ |e|_2 : |\tau'| \mid \lfloor\tau'\rfloor_a(\mathbf{r}_1,\mathbf{r}_2)$.
By i.h. on the first premise:
$|\Gamma| \mid \lfloor\Gamma\rfloor_a \vdash |e|_1 : |\tau| \sim |e|_2 : |\tau| \mid \lfloor\mathbb{T}_\ell(\tau)\rfloor_a(\mathbf{r}_1,\mathbf{r}_2)$ (1)
By i.h. on the second premise:
$|\Gamma|, x_1:|\tau|, x_2:|\tau| \mid \lfloor\Gamma\rfloor_a, \lfloor\tau\rfloor_a(x_1,x_2) \vdash |e'|_1 : |\tau'| \sim |e'|_2 : |\tau'| \mid \lfloor\tau'\rfloor_a(\mathbf{r}_1,\mathbf{r}_2)$ (2)
We consider two cases:
Subcase. $\ell \sqsubseteq a$. Here, $\lfloor\mathbb{T}_\ell(\tau)\rfloor_a(\mathbf{r}_1,\mathbf{r}_2) \triangleq \lfloor\tau\rfloor_a(\mathbf{r}_1,\mathbf{r}_2)$, so (1) can be rewritten to:
$|\Gamma| \mid \lfloor\Gamma\rfloor_a \vdash |e|_1 : |\tau| \sim |e|_2 : |\tau| \mid \lfloor\tau\rfloor_a(\mathbf{r}_1,\mathbf{r}_2)$ (3)
Applying rule ABS to (2) yields:
$|\Gamma| \mid \lfloor\Gamma\rfloor_a \vdash \lambda x_1.|e'|_1 : |\tau| \to |\tau'| \sim \lambda x_2.|e'|_2 : |\tau| \to |\tau'| \mid$
$\forall x_1 x_2.\lfloor\tau\rfloor_a(x_1,x_2) \Rightarrow \lfloor\tau'\rfloor_a(\mathbf{r}_1\ x_1, \mathbf{r}_2\ x_2)$ (4)
Applying rule APP to (4) and (3) yields:

$|\Gamma| \mid \lfloor\Gamma\rfloor_a \vdash (\lambda x_1.|e'|_1)\ |e|_1 : |\tau'| \sim (\lambda x_2.|e'|_2)\ |e|_2 : |\tau'| \mid \lfloor\tau'\rfloor_a(\mathbf{r}_1,\mathbf{r}_2)$
which is what we wanted to prove.
Subcase. $\ell \not\sqsubseteq a$. Here, $\lfloor\mathbb{T}_\ell(\tau)\rfloor_a(\mathbf{r}_1,\mathbf{r}_2) \triangleq \lfloor\tau\rfloor_a(\mathbf{r}_1,\mathbf{r}_2)$, so (1) can be rewritten to:
$|\Gamma| \mid \lfloor\Gamma\rfloor_a \vdash |e|_1 : |\tau| \sim |e|_2 : |\tau| \mid \top$ (5)
Applying rule ABS to (2) yields:
$|\Gamma| \mid \lfloor\Gamma\rfloor_a \vdash \lambda x_1.|e'|_1 : |\tau| \to |\tau'| \sim \lambda x_2.|e'|_2 : |\tau| \to |\tau'| \mid$
$\forall x_1 x_2.\lfloor\tau\rfloor_a(x_1,x_2) \Rightarrow \lfloor\tau'\rfloor_a(\mathbf{r}_1\ x_1,\mathbf{r}_2\ x_2)$
By Lemma 16 applied to the subcase assumption $\ell \not\sqsubseteq a$ and the premise $\tau' \searrow \ell$, we have $\lfloor\tau'\rfloor_a(\mathbf{r}_1\ x_1,\mathbf{r}_2\ x_2) \equiv \top$. So, by rule SUB:
$|\Gamma| \mid \lfloor\Gamma\rfloor_a \vdash \lambda x_1.|e'|_1 : |\tau| \to |\tau'| \sim \lambda x_2.|e'|_2 : |\tau| \to |\tau'| \mid \forall x_1 x_2.\lfloor\tau\rfloor_a(x_1,x_2) \Rightarrow \top$
Since $(\forall x_1 x_2.\lfloor\tau\rfloor_a(x_1,x_2) \Rightarrow \top) \equiv \top \equiv (\forall x_1,x_2.\top \Rightarrow \top)$, we can use SUB again to get:
$|\Gamma| \mid \lfloor\Gamma\rfloor_a \vdash \lambda x_1.|e'|_1 : |\tau| \to |\tau'| \sim \lambda x_2.|e'|_2 : |\tau| \to |\tau'| \mid \forall x_1,x_2.\top \Rightarrow \top$ (6)
Applying rule APP to (6) and (5) yields:
$|\Gamma| \mid \lfloor\Gamma\rfloor_a \vdash (\lambda x_1.|e'|_1)\ |e|_1 : |\tau'| \sim (\lambda x_2.|e'|_2)\ |e|_2 : |\tau'| \mid \top$
which is the same as our goal since $\lfloor\tau'\rfloor_a(\mathbf{r}_1,\mathbf{r}_2) \equiv \top$.
□

DCC's noninterference theorem is a corollary of this theorem and the soundness of RHOL in set theory.

7.4 Relational Cost

RelCost (Çiçek *et al.*, 2017) is a relational refinement type-and-effect system designed to reason about relative cost—the difference in the execution costs of two similar programs or of two runs of the same program on two different inputs. RelCost combines reasoning about the maximum and minimum costs of a single program with relational reasoning about the relative cost of two programs. This combination of reasoning styles is motivated by the following observation: if two programs are structurally similar, then relational reasoning can improve precision when computing their relative cost. And if this approach fails, one can still establish an upper bound on the relative cost by computing the difference of the maximum cost of one program and the minimum cost of the other.

Here, we show how a fragment of RelCost can be embedded into RHOL. Similar to what we did for DCC, to just convey the main intuition, we consider a fragment of RelCost excluding recursive functions. The syntax of RelCost is based on two sorts of types:

$$\begin{array}{rcll} A & ::= & \mathbb{N} \mid \mathsf{list}_A[n] \mid A \xrightarrow{\text{exec}(k,l)} A \mid \forall i \overset{\text{exec}(k,l)}{::} S.\,A & \text{(unary types)} \\ \tau & ::= & \mathbb{N}_r \mid \mathsf{list}_\tau[n]^\alpha \mid \tau \xrightarrow{\text{diff}(k)} \tau \mid \forall i \overset{\text{diff}(k)}{::} S.\,\tau \mid UA \mid \square\tau & \text{(relational types)} \end{array}$$

Unary types are used to type one program and they are mostly standard except for the effect annotation $\text{exec}(k,l)$ on arrow types and universally quantified types representing the min and max cost k and l of the body of the closure, respectively. Relational types ascribe two programs, so they are interpreted as pairs of expressions. In relational types, arrow types and universally quantified types have an effect annotation diff(k) representing the relative cost k of the two closures. Besides, the superscript α refines list types with the number of elements that can differ in two lists. The type UA is the weakest relation over elements of the unary type A, i.e. it can be used to type two arbitrary terms, while the type $\square\tau$ is the diagonal subrelation of τ, i.e. it can be used to type only two terms that are equal. There are two kinds of typing judgments, unary and relational:

$$\Delta;\Phi;\Omega \vdash_k^l t : A \qquad \Delta;\Phi;\Gamma \vdash t_1 \ominus t_2 \lesssim l : \tau$$

The unary judgment states that the execution cost of t is lower bounded by k and upper bounded by l, and the expression t has the unary type A. The relational judgment states that the relative cost of t_1 with respect to t_2 is upper bounded by l and the two expressions have the relational type τ. Here Ω is a unary type environment, Γ is a relational type environment, Δ is an environment for index variables and Φ for assumed constraints over the index terms. Figure 10 shows selected rules.

To embed RelCost in RHOL, we define a monadic-style cost-instrumented translation of RelCost types. The translation is given in two-steps: First, we define an erasure of cost and size information into simple types and then we define a cost-passing style translation of simple types with a value-translation and an expression-translation. The erasure function is defined as follows:

$$|\mathbb{N}| \triangleq |\mathbb{N}_r| \triangleq \mathbb{N} \qquad |\mathsf{list}_A[n]| \triangleq \mathsf{list}_{|A|} \qquad |\mathsf{list}_\tau[n]^\alpha| \triangleq \mathsf{list}_{|\tau|} \qquad |UA| \triangleq |A|$$

$$|\square\tau| \triangleq |\tau| \qquad |\forall i \overset{\mathrm{exec}(k,l)}{::} S.\, A| \triangleq \mathbb{N} \to |A| \qquad |\forall i \overset{\mathrm{diff}(k)}{::} S.\, \tau| \triangleq \mathbb{N} \to |\tau|$$

$$|A \xrightarrow{\mathrm{exec}(k,l)} B| \triangleq |A| \to |B| \qquad |\tau_1 \xrightarrow{\mathrm{diff}(k)} \tau_2| \triangleq |\tau_1| \to |\tau_2|$$

The cost-passing style translation of *simple* types is

$$(\!|\mathbb{N}|\!)_v \triangleq \mathbb{N} \qquad (\!|\mathsf{list}_A|\!)_v \triangleq \mathsf{list}_{(\!|A|\!)_v} \qquad (\!|A \to B|\!)_v \triangleq (\!|A|\!)_v \to (\!|B|\!)_e \qquad (\!|A|\!)_e \triangleq (\!|A|\!)_v \times \mathbb{N}$$

Guided by the translation of types above we can provide a cost-instrumented translation of simply-typed λ-expressions (Figure 11). This translation maps an expression of the simple type τ to an expression of type $\tau \times \mathbb{N}$, where the second component is the number of reduction steps under an eager, call-by-value reduction strategy (which is the semantics of RelCost). It is fairly easy to see that this translation preserves typability and that it counts steps accurately.

However, this translation forgets the cost and size information in types. To recover these, we define a HOL formula for every unary type. But, first, we define axiomatically a predicate $\mathsf{listU}(n,l,P)$ that captures size information about lists:

$$\forall l, P.\, \mathsf{listU}(0,l,P) \equiv l = []$$

$$\forall n, l, P.\, \mathsf{listU}(n+1,l,P) \equiv \exists w_1, w_2.\, l = w_1 :: w_2 \wedge P(w_1) \wedge \mathsf{listU}(n,w_2,P)$$

$$\text{VAR}\ \frac{\Omega(x) = A}{\Delta;\Phi_a;\Omega \vdash_0^0 x : A} \qquad \text{CONST}\ \frac{}{\Delta;\Phi_a;\Omega \vdash_0^0 \mathtt{n} : \text{int}}$$

$$\text{LAM}\ \frac{\Delta;\Phi_a;x : A_1,\Omega \vdash_k^l t : A_2}{\Delta;\Phi_a;\Omega \vdash_0^0 \lambda x.t : A_1 \xrightarrow{\text{exec}(k,l)} A_2}$$

$$\text{APP}\ \frac{\Delta;\Phi_a;\Omega \vdash_{k_1}^{l_1} t_1 : A_1 \xrightarrow{\text{exec}(k,l)} A_2 \qquad \Delta;\Phi_a;\Omega \vdash_{k_2}^{l_2} t_2 : A_1}{\Delta;\Phi_a;\Omega \vdash_{k_1+k_2+k+c_{app}}^{l_1+l_2+l+c_{app}} t_1\, t_2 : A_2}$$

$$\text{R-VAR}\ \frac{\Gamma(x) = \tau}{\Delta;\Phi_a;\Gamma \vdash x \ominus x \lesssim 0 : \tau} \qquad \text{R-CONST}\ \frac{}{\Delta;\Phi_a;\Gamma \vdash \mathtt{n} \ominus \mathtt{n} \lesssim 0 : \text{int}_r}$$

$$\text{R-CONS1}\ \frac{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim l_1 : \tau \qquad \Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim l_2 : \mathsf{list}_\tau[n]^\alpha}{\Delta;\Phi_a;\Gamma \vdash \text{cons}(t_1,t_2) \ominus \text{cons}(t_1',t_2') \lesssim l_1 + l_2 : \mathsf{list}_\tau[n+1]^{\alpha+1}}$$

$$\text{R-CONS2}\ \frac{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim l_1 : \Box\tau \qquad \Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim l_2 : \mathsf{list}_\tau[n]^\alpha}{\Delta;\Phi_a;\Gamma \vdash \text{cons}(t_1,t_2) \ominus \text{cons}(t_1',t_2') \lesssim l_1 + l_2 : \mathsf{list}_\tau[n+1]^{\alpha}}$$

$$\text{R-CASEL}\ \frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t \ominus t' \lesssim l : \mathsf{list}_\tau[n]^\alpha \qquad \Delta;\Phi_a \wedge n = 0;\Gamma \vdash t_1 \ominus t_1' \lesssim l' : \tau' \\ i,\Delta;\Phi_a \wedge n = i+1; h : \Box\tau, tl : \mathsf{list}_\tau[i]^\alpha, \Gamma \vdash t_2 \ominus t_2' \lesssim l' : \tau' \\ i,\beta,\Delta;\Phi_a \wedge n = i+1 \wedge \alpha = \beta + 1; h : \tau, tl : \mathsf{list}_\tau[i]^\beta, \Gamma \vdash t_2 \ominus t_2' \lesssim l' : \tau'\end{array}}{\Delta;\Phi_a;\Gamma \vdash \begin{array}{lr}\text{case } t \text{ of nil} & \to t_1 \\ \mid h :: tl & \to t_2\end{array} \ominus \begin{array}{lr}\text{case } t' \text{ of nil} & \to t_1' \\ \mid h :: tl & \to t_2'\end{array} \lesssim l + l' : \tau'}$$

$$\text{R-LAM}\ \frac{\Delta;\Phi_a;x : \tau_1,\Gamma \vdash t_1 \ominus t_2 \lesssim l : \tau_2}{\Delta;\Phi_a;\Gamma \vdash \lambda x.t_1 \ominus \lambda x.t_2 \lesssim 0 : \tau_1 \xrightarrow{\text{diff}(l)} \tau_2}$$

$$\text{R-APP}\ \frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_1' \lesssim l_1 : \tau_1 \xrightarrow{\text{diff}(l)} \tau_2 \\ \Delta;\Phi_a;\Gamma \vdash t_2 \ominus t_2' \lesssim l_2 : \tau_1\end{array}}{\Delta;\Phi_a;\Gamma \vdash t_1\, t_2 \ominus t_1'\, t_2' \lesssim l_1 + l_2 + l : \tau_2}$$

$$\text{R-iLAM}\ \frac{\begin{array}{c}i :: S, \Delta;\Phi_a;\Gamma \vdash t \ominus t' \lesssim l : \tau \\ i \notin \text{FIV}(\Phi_a;\Gamma)\end{array}}{\Delta;\Phi_a;\Gamma \vdash \Lambda t \ominus \Lambda t' \lesssim 0 : \forall i \overset{\text{diff}(l)}{::} S.\tau}$$

$$\text{R-iAPP}\ \frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t \ominus t' \lesssim l : \forall i \overset{\text{diff}(l')}{::} S.\tau \\ \Delta \vdash J : S\end{array}}{\Delta;\Phi_a;\Gamma \vdash t[\,] \ominus t'[\,] \lesssim l + l'[J/i] : \tau\{J/i\}}$$

$$\text{NOCHANGE}\ \frac{\begin{array}{c}\Delta;\Phi_a;\Gamma \vdash t \ominus t \lesssim l : \tau \\ \forall x \in dom(\Gamma).\ \ \Delta;\Phi_a \models \Gamma(x) \sqsubseteq \Box\Gamma(x)\end{array}}{\Delta;\Phi_a;\Gamma,\Gamma';\Omega \vdash t \ominus t \lesssim 0 : \Box\tau}$$

$$\text{SWITCH}\ \frac{\Delta;\Phi_a;\overline{\Gamma} \vdash_{k_1}^{l_1} t_1 : A \qquad \Delta;\Phi_a;\overline{\Gamma} \vdash_{k_2}^{l_2} t_2 : A}{\Delta;\Phi_a;\Gamma \vdash t_1 \ominus t_2 \lesssim l_1 - k_2 : U\,A}$$

Fig. 10. RelCost Unary and Relational Typing (Selected Rules)

$$(\!|x|\!) \triangleq (x,0) \qquad (\!|\lambda x.t|\!) \triangleq (\lambda x.(\!|t|\!),0) \qquad (\!|\Lambda.t|\!) \triangleq (\lambda\_.(\!|t|\!),0)$$

$$(\!|tu|\!) \triangleq \mathsf{let}\, x = (\!|t|\!)\, \mathsf{in}\, \mathsf{let}\, y = (\!|u|\!)\, \mathsf{in}\, \mathsf{let}\, z = \pi_1(x)\ \pi_1(y)\, \mathsf{in}\, (\pi_1(z), \pi_2(x)+\pi_2(y)+\pi_2(z)+c_{app})$$

$$(\!|t[]|\!) \triangleq \mathsf{let}\, x = (\!|t|\!)\, \mathsf{in}\, \mathsf{let}\, y = \pi_1(x)\ 0\, \mathsf{in}\, (\pi_1(y), \pi_2(x)+\pi_2(y)+c_{iapp}) \qquad (\!|\mathsf{nil}|\!) \triangleq (\mathsf{nil},0)$$

$$(\!|\mathsf{cons}(t_1,t_2)|\!) \triangleq \mathsf{let}\, x = (\!|t_1|\!)\, \mathsf{in}\, \mathsf{let}\, y = (\!|t_2|\!)\, \mathsf{in}\, (\pi_1(x) :: \pi_1(y), \pi_2(x)+\pi_2(y))$$

$$(\!|\ \text{case}\ t'\ \text{of nil}\ \to t_1' \mid h :: tl \to t_2'|\!) \triangleq$$

$$\begin{cases} \mathsf{let}\, x = (\!|t'|\!)\, \mathsf{in}\ \text{case}\ \pi_1(x)\ \text{of} \\ \text{nil}\ \to \mathsf{let}\, y = (\!|t_1'|\!)\, \mathsf{in}\, (\pi_1(y), \pi_2(x)+\pi_2(y)+c_{case}) \\ \ \mid h :: tl \to \mathsf{let}\, y = (\!|t_2'|\!)\, \mathsf{in}\, (\pi_1(y), \pi_2(x)+\pi_2(y)+c_{case}) \end{cases}$$

Fig. 11. Cost-instrumented translation of expressions.

We can now define a HOL formula inductively on unary types.

$$\lfloor \mathbb{N} \rfloor_v(x) \triangleq \top \qquad \lfloor \mathsf{list}_A[n] \rfloor_v(x) \triangleq \mathsf{listU}(n, x, \lfloor A \rfloor_v)$$

$$\lfloor A \xrightarrow{\text{exec}(k,l)} B \rfloor_v(x) \triangleq \forall y. \lfloor A \rfloor_v(y) \Rightarrow \lfloor B \rfloor_e^{k,l}(x\ y)$$

$$\lfloor \forall i \overset{\text{exec}(k,l)}{::} S. A \rfloor_v(x) \triangleq \forall y. \top \Rightarrow \forall i. \lfloor A \rfloor_e^{k,l}(x\ y)$$

$$\lfloor A \rfloor_e^{k,l}(x) \triangleq \lfloor A \rfloor_v(\pi_1 x) \wedge k \le \pi_2 x \le l$$

The type translation can also be extended to type environments: $(\!|\!|x_1 : A_1, \dots, x_n : A_n|\!|\!) = x_1 : (\!|\!|A_1|\!|\!)_v, \dots, x_n : (\!|\!|A_n|\!|\!)_v$ Similarly, we can associate to a type environment an HOL context that we can use to recover the cost and size information: $\lfloor x_1 : A_1, \dots, x_n : A_n \rfloor = \lfloor A_1 \rfloor_v(x_1), \dots, \lfloor A_n \rfloor_v(x_n)$. Now we can provide a cost-instrumented translation of unary judgments.

Theorem 18
If $\Delta; \Phi; \Omega \vdash_k^l t : A$, then: $(\!|\!|\Omega|\!|\!), \Delta \mid \Phi, \lfloor \Omega \rfloor \vdash (\!|t|\!) : (\!|\!|A|\!|\!)_e \mid \lfloor A \rfloor_e^{k,l}(\mathbf{r})$

Proof
By induction on the derivation of $\Delta; \Phi; \Omega \vdash_k^l t : A$. We show selected cases.

Case. $\dfrac{}{\Delta; \Phi_a; \Omega, x : A \vdash_0^0 x : A}$

We can conclude by the following derivation:

$$\dfrac{\dfrac{}{(\!|\!|\Omega|\!|\!), x : (\!|\!|A|\!|\!)_v, \Delta \mid \Phi_a, \lfloor \Omega \rfloor, \lfloor A \rfloor_v(x) \vdash x : (\!|\!|A|\!|\!)_v \mid \lfloor A \rfloor_v(\mathbf{r})}\ \textsc{Var} \qquad \dfrac{}{(\!|\!|\Omega|\!|\!), x : (\!|\!|A|\!|\!)_v, \Delta \mid \Phi_a, \lfloor \Omega \rfloor, \lfloor A \rfloor_v(x) \vdash 0 : \mathbb{N} \mid 0 \le \mathbf{r} \le 0}\ \textsc{Nat}}{(\!|\!|\Omega|\!|\!), x : (\!|\!|A|\!|\!)_v, \Delta \mid \Phi_a, \lfloor \Omega \rfloor, \lfloor A \rfloor_v(x) \vdash (x,0) : (\!|\!|A|\!|\!)_v \times \mathbb{N} \mid \lfloor A \rfloor_v(\pi_1 \mathbf{r}) \wedge 0 \le \pi_2 \mathbf{r} \le 0}\ \textsc{Pair-L}$$

where the additional proof condition that is needed for the [PAIR-L] rule can be easily proved in HOL.

Case. $\dfrac{}{\Delta; \Phi_a; \Omega \vdash_0^0 \mathtt{n} : \text{int}}$

Then we can conclude by the following derivation:

$$\dfrac{\dfrac{}{(\!|\Omega|\!), \Delta \mid \Phi_a, \lfloor\Omega\rfloor \vdash \mathbf{n} : \mathbb{N} \mid \top}\textsc{Nat} \qquad \dfrac{}{(\!|\Omega|\!), \Delta \mid \Phi_a, \lfloor\Omega\rfloor \vdash 0 : \mathbb{N} \mid 0 \leq \mathbf{r} \leq 0}\textsc{Nat}}{(\!|\Omega|\!), \Delta \mid \Phi_a, \lfloor\Omega\rfloor \vdash (\mathtt{n}, 0) : \mathbb{N} \times \mathbb{N} \mid 0 \leq \pi_2 \mathbf{r} \leq 0}\textsc{Pair-L}$$

where the additional proof condition that is needed for the [PAIR-L] rule can be easily proved in HOL.

Case. $\dfrac{\Delta; \Phi_a; x : A_1, \Omega \vdash_k^l t : A_2}{\Delta; \Phi_a; \Omega \vdash_0^0 \lambda x.t : A_1 \xrightarrow{\text{exec}(k,l)} A_2}$
By induction hypothesis we have $(\!|\Omega|\!), x : (\!|A_1|\!)_v, \Delta \mid \Phi, \lfloor\Omega\rfloor, \lfloor A_1\rfloor_v(x) \vdash (\!|t|\!) : (\!|A_2|\!)_e \mid \lfloor A\rfloor_e^{k,l}(\mathbf{r})$ and we can conclude by the following derivation:

$$\dfrac{\dfrac{\begin{array}{c}(\!|\Omega|\!), x : (\!|A_1|\!)_v, \Delta \mid \Phi, \lfloor\Omega\rfloor, \lfloor A_1\rfloor_v(x) \vdash \\ (\!|t|\!) : (\!|A_2|\!)_e \mid \lfloor A_2\rfloor_e^{k,l}(\mathbf{r})\end{array}}{\begin{array}{c}(\!|\Omega|\!), \Delta \mid \Phi, \lfloor\Omega\rfloor \vdash \lambda x.(\!|t|\!) : (\!|A_1|\!)_v \to (\!|A_2|\!)_e \mid \\ \forall x. \lfloor A_1\rfloor_v(x) \Rightarrow \lfloor A_2\rfloor_e^{k,l}(\mathbf{r}x)\end{array}}\textsc{Abs} \qquad (\!|\Omega|\!), \Delta \mid \Phi, \lfloor\Omega\rfloor \vdash 0 : \mathbb{N} \mid 0 \leq \mathbf{r} \leq 0}{\begin{array}{c}(\!|\Omega|\!), \Delta \mid \Phi, \lfloor\Omega\rfloor \vdash (\lambda x.(\!|t|\!), 0) : ((\!|A_1|\!)_v \to (\!|A_2|\!)_e) \times \mathbb{N} \mid \\ \forall x. \lfloor A_1\rfloor_v(x) \Rightarrow \lfloor A_2\rfloor_e^{k,l}((\pi_1 \mathbf{r})x) \wedge 0 \leq \pi_2 \mathbf{r} \leq 0\end{array}}\textsc{Pair-L}$$

where the additional proof condition that is needed for the [PAIR-L] rule can be easily proved in HOL.

Case $\dfrac{\Delta; \Phi_a; \Omega \vdash_{k_1}^{l_1} t_1 : A_1 \xrightarrow{\text{exec}(k,l)} A_2 \qquad \Delta; \Phi_a; \Omega \vdash_{k_2}^{l_2} t_2 : A_1}{\Delta; \Phi_a; \Omega \vdash_{k_1+k_2+k+c_{app}}^{l_1+l_2+l+c_{app}} t_1\, t_2 : A_2}$
By induction hypothesis and unfolding some some definitions we have

$$\begin{aligned}&(\!|\Omega|\!), \Delta \mid \Phi_a, \lfloor\Omega\rfloor \vdash (\!|t_1|\!) : ((\!|A_1|\!)_v \to ((\!|A_2|\!)_v \times \mathbb{N})) \times \mathbb{N} \mid \\ &\quad \forall h. \lfloor A_1\rfloor_v(h) \Rightarrow (\lfloor A_2\rfloor_v(\pi_1((\pi_1(\mathbf{r}))h)) \wedge k \leq \pi_2((\pi_1(\mathbf{r}))h) \leq l) \wedge k_1 \leq \pi_2(\mathbf{r}) \leq l_1\end{aligned}$$

and $(\!|\Omega|\!), \Delta \mid \Phi_a, \lfloor\Omega\rfloor \vdash (\!|t_2|\!) : (\!|A_1|\!)_v \times \mathbb{N} \mid \lfloor A_1\rfloor_v(\pi_1(\mathbf{r})) \wedge k_2 \leq \pi_2(\mathbf{r}) \leq l_2$. So, we can prove:

$$\begin{aligned}&(\!|\Omega|\!), \Delta \mid \Phi_a, \lfloor\Omega\rfloor \vdash \mathsf{let}\, x = (\!|t_1|\!)\, \mathsf{in}\, \mathsf{let}\, y = (\!|t_2|\!)\, \mathsf{in}\, \pi_1(x)\, \pi_1(y) : (\!|A_2|\!)_v \times \mathbb{N} \mid \\ &\qquad\qquad \lfloor A_2\rfloor_v(\pi_1(\mathbf{r})) \wedge k \leq \pi_2(\mathbf{r}) \leq l \wedge k_1 \leq \pi_2(x) \leq l_1 \wedge k_2 \leq \pi_2(y)\mathbf{r} \leq l_2\end{aligned}$$

This combined with the definition of the cost-passing translation $(\!|t_1\, t_2|\!) \triangleq \mathsf{let}\, x = (\!|t_1|\!)\, \mathsf{in}\, \mathsf{let}\, y = (\!|t_2|\!)\, \mathsf{in}\, \mathsf{let}\, z = \pi_1(x)\ \pi_1(y)\, \mathsf{in}\, (\pi_1(z), \pi_2(x) + \pi_2(y) + \pi_2(z) + c_{app})$ allows us to prove as required the following:

$$\begin{gathered}(\!|\Omega|\!), \Delta \mid \Phi_a, \lfloor\Omega\rfloor \vdash (\!|t_1\, t_2|\!) : (\!|A_2|\!)_v \times \mathbb{N} \mid \\ \lfloor A_2\rfloor_v(\pi_1(\mathbf{r})) \wedge k + k_1 + k_2 + c_{app} \leq \pi_2(\mathbf{r}) \leq l + l_1 + l_2 + c_{app}.\end{gathered}$$

□

For the embedding of cost and size information in the relational case we first define a predicate $\mathsf{listR}(n,l_1,l_2,a,P)$ in HOL axiomatically:

$$\forall l_1,l_2,a,P.\, \mathsf{listR}(0,l_1,l_2,a,P) \equiv l_1 = l_2 = [] \qquad \forall n,l_1,l_2,a,P.\, \mathsf{listR}(n+1,l_1,l_2,a,P) \equiv$$

$$\begin{array}{c}\exists w_1,z_1,w_2,z_2.\, l_1 = w_1 :: w_2 \wedge l_2 = z_1 :: z_2 \wedge P(w_1,z_1) \wedge \\ (((w_1 = z_1) \wedge \mathsf{listR}(n,w_2,z_2,a,P)) \vee \\ (a > 0 \wedge \exists b.\, a = b+1 \wedge \mathsf{listR}(n,w_2,z_2,b,P)))\end{array}$$

Let $\overline{\tau}$ denote RelCost's erasure of the binary type τ to a unary type.[2] This erasure maps $\mathsf{list}_\tau[n]^\alpha$ to $\mathsf{list}_{\overline{\tau}}[n]$, $\tau \xrightarrow{\text{diff}(l)} \sigma$ to $\overline{\tau} \xrightarrow{\text{exec}(0,\infty)} \overline{\sigma}$, etc. Next, we define HOL formulas for the binary types.

$$\llbracket \mathbb{N} \rrbracket_v(x,y) \triangleq x = y \qquad\qquad \llbracket UA \rrbracket_v(x,y) \triangleq \lfloor A \rfloor_v(x) \wedge \lfloor A \rfloor_v(y)$$

$$\llbracket \square \tau \rrbracket_v(x,y) \triangleq (x = y) \wedge (\llbracket \tau \rrbracket_v(x,y))$$

$$\llbracket \tau \xrightarrow{\text{diff}(l)} \sigma \rrbracket_v(x,y) \triangleq \left\{ \begin{array}{l} \lfloor \overline{\tau} \xrightarrow{\text{exec}(0,\infty)} \overline{\sigma} \rfloor_v(x) \wedge \lfloor \overline{\tau} \xrightarrow{\text{exec}(0,\infty)} \overline{\sigma} \rfloor_v(y) \wedge \\ (\forall z_1,z_2. \llbracket \tau \rrbracket_v(z_1,z_2) \Rightarrow \llbracket \sigma \rrbracket_e^l(x\ z_1, y\ z_2)) \end{array} \right.$$

$$\llbracket \forall i \overset{\text{diff}(l)}{::} S.\, \tau \rrbracket_v(x,y) \triangleq \left\{ \begin{array}{l} \lfloor \forall i \overset{\text{exec}(0,\infty)}{::} S.\, \overline{\tau} \rfloor_v(x) \wedge \lfloor \forall i \overset{\text{exec}(0,\infty)}{::} S.\, \overline{\tau} \rfloor_v(y) \wedge \\ (\forall z_1 z_2. \top \Rightarrow \forall i. \llbracket \tau \rrbracket_e^l(x\ z_1, y\ z_2)) \end{array} \right.$$

$$\llbracket \mathsf{list}_\tau[n]^\alpha \rrbracket_v(x,y) \triangleq \mathsf{listR}(n,x,y,\alpha,\llbracket \tau \rrbracket_v)$$

$$\llbracket \tau \rrbracket_e^l(x,y) \triangleq \llbracket \tau \rrbracket_v(\pi_1 x, \pi_1 y) \wedge (\pi_2 x - \pi_2 y \leq l)$$

The type translation can also be extended to relational type environments pointwise: $\|x^1 : \tau_1, \dots, x^n : \tau_n\| \triangleq x_1^1 : (\!|\tau_1|\!)_v, x_2^1 : (\!|\tau_1|\!)_v, \dots, x_1^n : (\!|\tau_n|\!)_v, x_2^n : (\!|\tau_n|\!)_v$ We also need to derive from a type relational environment an HOL context that remembers the cost and size information: $\llbracket x^1 : \tau_1, \dots, x^n : \tau_n \rrbracket \triangleq \llbracket \tau_1 \rrbracket_v(x_1^1, x_2^1), \dots, \llbracket \tau_n \rrbracket_v(x_1^n, x_2^n)$. Now we can provide the translation of relational judgments.

Theorem 19
If $\Delta;\Phi;\Gamma \vdash t_1 \ominus t_2 \lesssim l : \tau$, then:

$$\|\Gamma\|, \Delta \mid \Phi, \llbracket \Gamma \rrbracket \vdash (\!|t_1|\!)_1 : (\!|\tau|\!)_e \sim (\!|t_2|\!)_2 : (\!|\tau|\!)_e \mid \llbracket \tau \rrbracket_e^l(\mathbf{r}_1, \mathbf{r}_2),$$

where $(\!|t_i|\!)_j$ is a copy of t_i where each variable x is replaced by a variable x_j for $j \in \{1,2\}$.

To prove Theorem 19, we need three lemmas.

Lemma 20
Suppose $\Delta;\Phi \vdash \tau\ \mathsf{wf}$.[3] Then, the following hold:

[2] In RelCost, this erasure is written $|\tau|$. We use a different notation to avoid confusion with our own erasure function from RelCost's types to simple types.

[3] This judgment simply means that τ is well-formed in the context $\Delta;\Phi$. It is defined in the original RelCost paper (Çiçek *et al.*, 2017).

1. $\Delta \mid \Phi \vdash \forall xy. \llbracket \tau \rrbracket_v(x,y) \Rightarrow \lfloor \overline{\tau} \rfloor_v(x) \wedge \lfloor \overline{\tau} \rfloor_v(y)$
2. $\Delta \mid \Phi \vdash \forall xy. \llbracket \tau \rrbracket_e^t(x,y) \Rightarrow \lfloor \overline{\tau} \rfloor_e^{0,\infty}(x) \wedge \lfloor \overline{\tau} \rfloor_e^{0,\infty}(y)$

Also, (3) $\llbracket \Gamma \rrbracket \Rightarrow \lfloor \overline{\Gamma}_1 \rfloor \wedge \lfloor \overline{\Gamma}_2 \rfloor$ where $\overline{\Gamma}_1$ and $\overline{\Gamma}_2$ are obtained by replacing each variable x in $\overline{\Gamma}$ with x_1 and x_2, respectively.

Proof
(1) and (2) follow by a simultaneous induction on the given judgment. (3) follows immediately from (1). □

Lemma 21
If $\Delta;\Phi_a;\Gamma \vdash e_1 \ominus e_2 \lesssim t : \tau$ in RelCost, then $\Delta;\Phi;\overline{\Gamma} \vdash_0^\infty e_i : \overline{\tau}$ for $i \in \{1,2\}$ in RelCost.

Proof
By induction on the given derivation. □

Lemma 22
If $\Delta;\Phi \models \tau_1 \sqsubseteq \tau_2$, then $\Delta;\Phi \vdash \forall xy. \llbracket \tau_1 \rrbracket_v(x,y) \Rightarrow \llbracket \tau_2 \rrbracket_v(x,y)$.

Proof
By induction on the given derivation of $\Delta;\Phi \models \tau_1 \sqsubseteq \tau_2$. □

Proof of Theorem 19
The proof is by induction on the given derivation of $\Delta;\Phi;\Gamma \vdash t_1 \ominus t_2 \lesssim k : \tau$. We show only a few representative cases here.

$$\textbf{Case:} \quad \frac{i :: S, \Delta;\Phi_a;\Gamma \vdash e \ominus e' \lesssim t : \tau \qquad i \notin \mathrm{FIV}(\Phi_a;\Gamma)}{\Delta;\Phi_a;\Gamma \vdash \Lambda e \ominus \Lambda e' \lesssim 0 : \forall i \overset{\mathrm{diff}(t)}{::} S.\tau} \textsc{r-iLam}$$

To show: $\|\Gamma\|, \Delta \mid \Phi_a, \llbracket \Gamma \rrbracket \vdash (\lambda\_.(\!|e|\!)_1, 0) : (\mathbb{N} \to (\!|\|\tau\||\!)_e) \times \mathbb{N} \sim (\lambda\_.(\!|e'|\!)_2, 0) : (\mathbb{N} \to (\!|\|\tau\||\!)_e) \times \mathbb{N} \mid \llbracket \forall i \overset{\mathrm{diff}(t)}{::} S.\tau \rrbracket_e^0(\mathbf{r}_1, \mathbf{r}_2)$.

Expand $\llbracket \forall i \overset{\mathrm{diff}(t)}{::} S.\tau \rrbracket_e^0(\mathbf{r}_1, \mathbf{r}_2)$ to $\llbracket \forall i \overset{\mathrm{diff}(t)}{::} S.\tau \rrbracket_v(\pi_1\ \mathbf{r}_1, \pi_1\ \mathbf{r}_2) \wedge \pi_2 \mathbf{r}_1 - \pi_2\ \mathbf{r}_2 \leq 0$, and apply the rule [PAIR] to reduce to two proof obligations:

(A) $\|\Gamma\|, \Delta \mid \Phi_a, \llbracket \Gamma \rrbracket \vdash \lambda\_.(\!|e|\!)_1 : \mathbb{N} \to (\!|\|\tau\||\!)_e \sim \lambda\_.(\!|e'|\!)_2 : \mathbb{N} \to (\!|\|\tau\||\!)_e \mid \llbracket \forall i \overset{\mathrm{diff}(t)}{::} S.\tau \rrbracket_v(\mathbf{r}_1, \mathbf{r}_2)$

(B) $\|\Gamma\|, \Delta \mid \Phi_a, \llbracket \Gamma \rrbracket \vdash 0 : \mathbb{N} \sim 0 : \mathbb{N} \mid \mathbf{r}_1 - \mathbf{r}_2 \leq 0$

(B) follows immediately by rule [ZERO]. To prove (A), we start by expanding $\llbracket \forall i \overset{\mathrm{diff}(t)}{::} S.\tau \rrbracket_v(\mathbf{r}_1, \mathbf{r}_2)$ and apply rule [$\wedge_\mathsf{I}$]. We get three proof obligations.

(C) $\|\Gamma\|, \Delta \mid \Phi_a, \llbracket \Gamma \rrbracket \vdash \lambda\_.(\!|e|\!)_1 : \mathbb{N} \to (\!|\|\tau\||\!)_e \sim \lambda\_.(\!|e'|\!)_2 : \mathbb{N} \to (\!|\|\tau\||\!)_e \mid \lfloor \forall i \overset{\mathrm{exec}(0,\infty)}{::} S.\overline{\tau} \rfloor_v(\mathbf{r}_1)$

(D) $\|\Gamma\|, \Delta \mid \Phi_a, \llbracket \Gamma \rrbracket \vdash \lambda\_.(\!|e|\!)_1 : \mathbb{N} \to (\!|\|\tau\||\!)_e \sim \lambda\_.(\!|e'|\!)_2 : \mathbb{N} \to (\!|\|\tau\||\!)_e \mid \lfloor \forall i \overset{\mathrm{exec}(0,\infty)}{::} S.\overline{\tau} \rfloor_v(\mathbf{r}_2)$

(E) $\|\Gamma\|, \Delta \mid \Phi_a, \llbracket \Gamma \rrbracket \vdash \lambda\_.(\!|e|\!)_1 : \mathbb{N} \to (\!|\|\tau\||\!)_e \sim \lambda\_.(\!|e'|\!)_2 : \mathbb{N} \to (\!|\|\tau\||\!)_e \mid \forall z_1 z_2. \top \Rightarrow \forall i. \llbracket \tau \rrbracket_e^t(\mathbf{r}_1\ z_1, \mathbf{r}_2\ z_2)$

To prove (C), apply Lemma 34 to the given derivation (not just the premise), obtaining a RelCost derivation for $\Delta;\Phi_a;\overline{\Gamma} \vdash_0^\infty \Lambda e : (\forall i \overset{\mathrm{exec}(0,\infty)}{::} S.\overline{\tau})$. Applying Theorem 18 to this yields $(\!|\overline{\Gamma}|\!), \Delta \mid \Phi_a, \lfloor \overline{\Gamma} \rfloor \vdash (\lambda\_.(\!|e|\!), 0) : (\mathbb{N} \to (\!|\|\overline{\tau}\||\!)_e) \times \mathbb{N} \mid \lfloor \forall i \overset{\mathrm{exec}(0,\infty)}{::}$

$S.\overline{\tau}\rfloor_e^{0,\infty}(\mathbf{r})$ in UHOL, which is the same as $(\!|\overline{\Gamma}|\!), \Delta \mid \Phi_a, \lfloor\overline{\Gamma}\rfloor \vdash (\lambda\_.(\!|e|\!), 0) : (\mathbb{N} \to (\!|\|\overline{\tau}\||\!)_e) \times \mathbb{N} \mid \lfloor\forall i \overset{\mathrm{exec}(0,\infty)}{::} S.\overline{\tau}\rfloor_v(\pi_1\ \mathbf{r}) \wedge 0 \leq \pi_2\ \mathbf{r} \leq \infty$. Applying rule [PROJ$_1$], we get $(\!|\overline{\Gamma}|\!), \Delta \mid \Phi_a, \lfloor\overline{\Gamma}\rfloor \vdash \pi_1(\lambda\_.(\!|e|\!), 0) : \mathbb{N} \to (\!|\|\overline{\tau}\||\!)_e \mid \lfloor\forall i \overset{\mathrm{exec}(0,\infty)}{::} S.\overline{\tau}\rfloor_v(\mathbf{r})$. By subject conversion, $(\!|\overline{\Gamma}|\!), \Delta \mid \Phi_a, \lfloor\overline{\Gamma}\rfloor \vdash \lambda\_.(\!|e|\!) : \mathbb{N} \to (\!|\|\overline{\tau}\||\!)_e \mid \lfloor\forall i \overset{\mathrm{exec}(0,\infty)}{::} S.\overline{\tau}\rfloor_v(\mathbf{r})$. Renaming variables, we get $(\!|\overline{\Gamma}|\!)_1, \Delta \mid \Phi_a, \lfloor\overline{\Gamma}_1\rfloor \vdash \lambda\_.(\!|e|\!)_1 : \mathbb{N} \to (\!|\|\overline{\tau}\||\!)_e \mid \lfloor\forall i \overset{\mathrm{exec}(0,\infty)}{::} S.\overline{\tau}\rfloor_v(\mathbf{r})$.

Now note that by definition, $\|\Gamma\| \supseteq (\!|\overline{\Gamma}|\!)_1$ and by Lemma 33(3), $\llbracket\Gamma\rrbracket \Rightarrow \lfloor\overline{\Gamma}_1\rfloor$. Hence, we also get $\|\Gamma\|, \Delta \mid \Phi_a, \llbracket\Gamma\rrbracket \vdash \lambda\_.(\!|e|\!)_1 : \mathbb{N} \to (\!|\|\overline{\tau}\||\!)_e \mid \lfloor\forall i \overset{\mathrm{exec}(0,\infty)}{::} S.\overline{\tau}\rfloor_v(\mathbf{r})$. (C) follows immediately by rule [UHOL-L].

(D) has a similar proof.

To prove (E), apply the rule [ABS], getting the obligation:
$\|\Gamma\|, \Delta, z_1, z_2 : \mathbb{N} \mid \Phi_a, \llbracket\Gamma\rrbracket \vdash (\!|e|\!)_1 : (\!|\|\tau\||\!)_e \sim (\!|e'|\!)_2 : (\!|\|\tau\||\!)_e \mid \forall i. \llbracket\tau\rrbracket_e^t(\mathbf{r}_1, \mathbf{r}_2)$
Since z_1, z_2 do not appear anywhere else, we can strengthen the context to remove them, thus reducing to: $\|\Gamma\|, \Delta \mid \Phi_a, \llbracket\Gamma\rrbracket \vdash (\!|e|\!)_1 : (\!|\|\tau\||\!)_e \sim (\!|e'|\!)_2 : (\!|\|\tau\||\!)_e \mid \forall i. \llbracket\tau\rrbracket_e^t(\mathbf{r}_1, \mathbf{r}_2)$
Next, we transpose to HOL using Theorem 6. We get the obligation:
$\|\Gamma\|, \Delta \mid \Phi_a, \llbracket\Gamma\rrbracket \vdash \forall i. \llbracket\tau\rrbracket_e^t((\!|e|\!)_1, (\!|e'|\!)_2)$
This is equivalent to:
$\|\Gamma\|, \Delta, i : S \mid \Phi_a, \llbracket\Gamma\rrbracket \vdash \llbracket\tau\rrbracket_e^t((\!|e|\!)_1, (\!|e'|\!)_2)$
The last statement follows immediately from i.h. on the premise, followed by transposition to HOL using Theorem 6.

Case: $$\frac{\Delta; \Phi_a; \Gamma \vdash e \ominus e \lesssim t : \tau \qquad \forall x \in dom(\Gamma).\ \ \Delta; \Phi_a \models \Gamma(x) \sqsubseteq \Box\Gamma(x)}{\Delta; \Phi_a; \Gamma, \Gamma'; \Omega \vdash e \ominus e \lesssim 0 : \Box\tau}\ \textsc{nochange}$$
To show: $\|\Gamma\|, \Delta \mid \Phi_a, \llbracket\Gamma\rrbracket \vdash (\!|e|\!)_1 : (\!|\|\tau\||\!)_e \sim (\!|e|\!)_2 : (\!|\|\tau\||\!)_e \mid \llbracket\Box\tau\rrbracket_e^0(\mathbf{r}_1, \mathbf{r}_2)$.
Expanding the definition of $\llbracket\Box\tau\rrbracket_e^0$, this is equivalent to:
$\|\Gamma\|, \Delta \mid \Phi_a, \llbracket\Gamma\rrbracket \vdash (\!|e|\!)_1 : (\!|\|\tau\||\!)_e \sim (\!|e|\!)_2 : (\!|\|\tau\||\!)_e \mid \llbracket\tau\rrbracket_v(\pi_1\ \mathbf{r}_1, \pi_2\ \mathbf{r}_2) \wedge (\pi_1\ \mathbf{r}_1 = \pi_1\ \mathbf{r}_2) \wedge (\pi_2\ \mathbf{r}_1 - \pi_2\ \mathbf{r}_2 \leq 0)$
Using rule [$\wedge_I$], we reduce this to two obligations:
(A) $\|\Gamma\|, \Delta \mid \Phi_a, \llbracket\Gamma\rrbracket \vdash (\!|e|\!)_1 : (\!|\|\tau\||\!)_e \sim (\!|e|\!)_2 : (\!|\|\tau\||\!)_e \mid \llbracket\tau\rrbracket_v(\pi_1\ \mathbf{r}_1, \pi_2\ \mathbf{r}_2)$
(B) $\|\Gamma\|, \Delta \mid \Phi_a, \llbracket\Gamma\rrbracket \vdash (\!|e|\!)_1 : (\!|\|\tau\||\!)_e \sim (\!|e|\!)_2 : (\!|\|\tau\||\!)_e \mid (\pi_1\ \mathbf{r}_1 = \pi_1\ \mathbf{r}_2) \wedge (\pi_2\ \mathbf{r}_1 - \pi_2\ \mathbf{r}_2 \leq 0)$

By i.h. on the first premise,
$\|\Gamma\|, \Delta \mid \Phi_a, \llbracket\Gamma\rrbracket \vdash (\!|e|\!)_1 : (\!|\|\tau\||\!)_e \sim (\!|e|\!)_2 : (\!|\|\tau\||\!)_e \mid \llbracket\tau\rrbracket_v(\pi_1\ \mathbf{r}_1, \pi_2\ \mathbf{r}_2) \wedge (\pi_2\ \mathbf{r}_1 - \pi_2\ \mathbf{r}_2 \leq t)$
By rule [SUB],
$\|\Gamma\|, \Delta \mid \Phi_a, \llbracket\Gamma\rrbracket \vdash (\!|e|\!)_1 : (\!|\|\tau\||\!)_e \sim (\!|e|\!)_2 : (\!|\|\tau\||\!)_e \mid \llbracket\tau\rrbracket_v(\pi_1\ \mathbf{r}_1, \pi_2\ \mathbf{r}_2)$
which is the same as (A).

To prove (B), apply Lemma 35 to the second premise to get for every $x \in dom(\Gamma)$ that $\Delta \mid \Phi_a \vdash \llbracket\Gamma(x)\rrbracket_v(x_1, x_2) \Rightarrow \llbracket\Box\Gamma(x)\rrbracket_v(x_1, x_2)$. Since $\llbracket\Box\Gamma(x)\rrbracket_v(x_1, x_2) \Rightarrow x_1 = x_2$ and from $\llbracket\Gamma\rrbracket$ we know that $\llbracket\Gamma(x)\rrbracket_v(x_1, x_2)$, it follows that $\|\Gamma\|, \Delta \mid \Phi_a, \llbracket\Gamma\rrbracket \vdash x_1 = x_2$. Since this holds for every $x \in dom(\Gamma)$, it follows immediately that $\|\Gamma\|, \Delta \mid \Phi_a, \llbracket\Gamma\rrbracket \vdash (\!|e|\!)_1 = (\!|e|\!)_2$. By Theorem 6, $\|\Gamma\|, \Delta \mid \Phi_a, \llbracket\Gamma\rrbracket \vdash (\!|e|\!)_1 : (\!|\|\tau\||\!)_e \sim (\!|e|\!)_2 : (\!|\|\tau\||\!)_e \mid \mathbf{r}_1 = \mathbf{r}_2$. (B) follows immediately by rule [SUB].

□

RelCost's type-soundness theorem can be derived from Theorem 19 and the soundness of RHOL in set theory.

8 Examples

We present some illustrative examples to show how RHOL's rules work in practice. Our first example shows the functional equivalence of two recursive functions that are synchronous—they perform the same number of recursive calls. The second example shows the equivalence of two asynchronous recursive functions. The third example shows a sensitivity property of sorting. Finally, our fourth example illustrates reasoning about the relative cost of two programs, using an encoding similar to that of RelCost, but the example cannot be verified in RelCost itself.

Notational simplifications Throughout this section, we often omit types and typing contexts when they are clear. We also apply the [SUB] rule implicitly in some places, e.g., to change the assertion of a function from $\forall x.\phi$ to $\forall x.\top \Rightarrow \phi$ so that we can apply [ABS], to rearrange the order of quantified variables; or to pull quantifiers outwards when there is no variable capture.

8.1 First Example: Factorial

Expanding on Section 3, we show that the two following standard implementations of factorial, with and without an accumulator, are functionally equivalent:

$$\text{fact}_1 \triangleq \text{letrec } f_1\ n_1 = \text{case } n_1 \textsf{ of } 0 \mapsto 1; S \mapsto \lambda x_1.(S\ x_1) * (f_1\ x_1)$$
$$\text{fact}_2 \triangleq \text{letrec } f_2\ n_2 = \lambda a.\text{case } n_2 \textsf{ of } 0 \mapsto a; S \mapsto \lambda x_2.f_2\ x_2\ ((S\ x_2) * a)$$

Our goal is to prove that the result is the same on both implementations after scaling the result on the first one by the accumulator. In RHOL, this is expressed by the following judgment (with empty contexts):

$$\text{fact}_1 : \mathbb{N} \to \mathbb{N} \sim \text{fact}_2 : \mathbb{N} \to \mathbb{N} \to \mathbb{N} \mid \forall n_1 n_2. n_1 = n_2 \Rightarrow \forall a.(\mathbf{r}_1\ n_1) * a = \mathbf{r}_2\ n_2\ a$$

The proof starts by applying [LETREC], which has the following main premise:

$$\Psi \vdash \begin{array}{l}\text{case } n_1 \text{ of}\\ 0 \mapsto 1;\\ S \mapsto \lambda x_1.(S\ x_1) * (f_1\ x_1)\end{array} \sim \begin{array}{l}\lambda a.\ \text{case } n_2 \text{ of}\\ \quad 0 \mapsto a;\\ \quad S \mapsto \lambda x_2.f_2\ x_2\ ((S\ x_2) * a)\end{array} \mid \forall a.\mathbf{r}_1 * a = \mathbf{r}_2\ a$$

where $\Psi \triangleq n_1 = n_2, \forall y_1 y_2.(y_1, y_2) < (n_1, n_2) \Rightarrow y_1 = y_2 \Rightarrow \forall a.(f_1\ y_1) * a = f_2\ y_2\ a$ asserts the inductive hypothesis and the equality between the arguments.

To prove this premise, we start by applying the one-sided [ABS-R] rule, with a trivial condition on a. Then we can apply a two-sided [CASE] rule, which has 3 premises. The first one asserts that the same branch is taken on both sides. The other two consider respectively the zero and the successor case. Since the branching is synchronous we do not need to consider the crossed cases:

1. $\Psi \vdash n_1 = 0 \Leftrightarrow n_2 = 0$

2. $\Psi, n_1 = 0, n_2 = 0 \vdash 1 \sim a \mid \mathbf{r}_1 * a = \mathbf{r}_2$
3. $\Psi \vdash \lambda x_1.(S\ x_1) * (f_1\ x_1) \sim \lambda x_2.f_2\ x_2\ ((S\ x_2) * a) \mid \forall x_1 x_2. n_1 = S\ x_1 \Rightarrow n_2 = S\ x_2 \Rightarrow (\mathbf{r}_1\ x_1) * a = \mathbf{r}_2\ x_2$

Premise 1 is a direct consequence of the assertion $n_1 = n_2$ in Ψ. Premise 2 is a trivial arithmetic identity which can be proven in HOL (using rule SUB or by invoking Theorem 6). To prove premise 3, we first apply the (two-sided) [ABS] rule, which leaves the following proof obligation:

$$\Psi, n_1 = S\ x_1, n_2 = S\ x_2 \vdash (S\ x_1) * (f_1\ x_1) \sim f_2\ x_2\ ((S\ x_2) * a) \mid \mathbf{r}_1 * a = \mathbf{r}_2$$

This is proven in HOL by instantiating the inductive hypothesis in Ψ with $y_1 \mapsto x_1, y_2 \mapsto x_2, a \mapsto (S\ x_1) * a$.

8.2 Second Example: Take and Map

This example establishes the equivalence of two programs that compute the same result, but using different number of recursive calls. Consider the following function *take* that takes a list l and a natural number n and returns the first n elements of the list (or the whole list if its length is less than n).

$$\begin{aligned} take \triangleq \text{letrec } f_1\ l_1 = \lambda n_1.\text{case } l_1 \text{ of } & [] \mapsto [] \\ & \_ :: \_ \mapsto \lambda h_1 t_1.\ \text{case } n_1 \text{ of } 1 \mapsto [] \\ & \qquad S \mapsto \lambda y_1.h_1 :: (f_1\ t_1\ y_1) \end{aligned}$$

Next, consider the standard function *map* that applies a function g to every element of a list l pointwise.

$$\begin{aligned} map \triangleq \text{letrec } f_2\ l_2 = \lambda g_2.\ \text{case } l_2 \text{ of } & [] \mapsto \quad [] \\ & ;\_ :: \_ \mapsto \quad \lambda h_2 t_2.(g_2\ h_2) :: (f_2\ t_2\ g_2) \end{aligned}$$

Intuitively, it should be clear that for all g, n, l, $map\ (take\ l\ n)\ g = take\ (map\ l\ g)\ n$ (mapping g over the first n elements of the list is the same as mapping g over the whole list and then taking the first n elements). However, the computations on the two sides of the equality are very different: For a list l of length more than n, $map\ (take\ l\ n)\ g$ only examines the first n elements, whereas $take\ (map\ l\ g)\ n$ traverses the whole list. In the following we formalize this property in RHOL (Theorem 23) and outline the high-level idea of the proof. The full proof is in the appendix (Section D.3).

Theorem 23
$l_1, l_2 : \mathsf{list}_{\mathbb{N}}, n_1, n_2 : \mathbb{N}, g_1, g_2 : \mathbb{N} \to \mathbb{N} \mid l_1 = l_2, n_1 = n_2, g_1 = g_2 \vdash$
$map\ (take\ l_1\ n_1)\ g_1 : \mathsf{list}_{\mathbb{N}} \sim take\ (map\ l_2\ g_2)\ n_2 : \mathsf{list}_{\mathbb{N}} \mid \mathbf{r}_1 = \mathbf{r}_2$

Proof idea
Since the two sides make an unequal number of recursive calls, we need to reason asynchronously on the two sides (specifically, we use the rule [LLCASE-A]). However, equality cannot be established inductively with asynchronous reasoning: If two function applications are to be shown equal, and a recursion step is taken in only one of them, then the induction hypothesis cannot be applied. So, we strengthen the

induction hypothesis, replacing the assertion $\mathbf{r}_1 = \mathbf{r}_2$ in the theorem statement with $\mathbf{r}_1 \sqsubseteq \mathbf{r}_2 \wedge |\mathbf{r}_1| = \mathsf{min}(n_1, |l_1|) \wedge |\mathbf{r}_2| = \mathsf{min}(n_2, |l_2|)$ where $\sqsubseteq$ denotes the prefix ordering on lists and $|\cdot|$ is the list length function. This assertion implies $\mathbf{r}_1 = \mathbf{r}_2$ and can be established inductively. The full proof is in the appendix, but at a high-level, the proof requires proving two judgments, one for the inner map-take pair and another for the outer one:

- $\Psi \vdash take\ l_1\ n_1 \sim map\ l_2\ g_2 \mid \mathbf{r}_1 \sqsubseteq_{g_2} \mathbf{r}_2$
- $\Psi \vdash map \sim take \mid \forall m_1 m_2 . m_1 \sqsubseteq_{g_2} m_2 \Rightarrow$
  $(\forall g_1 . g_1 = g_2 \Rightarrow \forall x_2 . x_2 \geq |m_1| \Rightarrow (\mathbf{r}_1\ m_1\ g_1) \sqsubseteq (\mathbf{r}_2\ m_2\ x_2))$

where $m_1 \sqsubseteq_g m_2$ is an axiomatically defined predicate equivalent to $(\text{map } m_1\ g) \sqsubseteq m_2$ and Ψ are the assumptions in the statement of the theorem (in particular, $l_1 = l_2$). The proof of the first premise proceeds by an analysis of *map* using synchronous rules. For the second premise, after applying [LETREC] we apply the asynchronous [$\mathsf{LLCASE\text{-}A}$] rule, and then prove the following premises:

1. $\Psi, \Phi, x_2 \geq |m_1|, g_1 = g_2, m_1 = [], m_2 = [] \vdash [] \sim [] \mid \mathbf{r}_1 \sqsubseteq \mathbf{r}_2$
2. $\Psi, \Phi, x_2 \geq |m_1|, g_1 = g_2, m_1 = [] \vdash [] \sim \lambda h_2 t_2 . \text{case } x_2 \text{ of } 0 \mapsto []; S \mapsto \lambda y_2 . h_2 :: f_2\ t_2\ y_2 \mid$
   $\forall h_2 t_2 . m_2 = h_2 :: t_2 \Rightarrow \mathbf{r}_1 \sqsubseteq (\mathbf{r}_2\ h_2\ t_2)$
3. $\Psi, \Phi, x_2 \geq |m_1|, g_1 = g_2, m_2 = [] \vdash \lambda h_1 t_1 . (g_1\ h_1) :: (f_1\ t_1\ g_1) \sim [] \mid \forall h_1 t_1 . m_1 = h_1 :: t_1 \Rightarrow (\mathbf{r}_1\ h_1\ t_1) \sqsubseteq \mathbf{r}_2$
4. $\Psi, \Phi, x_2 \geq |m_1|, g_1 = g_2 \vdash \lambda h_1 t_1 . (g_1\ h_1) :: (f_1\ t_1\ g_1) \sim \lambda h_2 t_2 . \text{case } x_2 \text{ of } 0 \mapsto []; S \mapsto \lambda y_2 . h_2 :: f_2\ t_2\ y_2 \mid \forall h_1 t_1 h_2 t_2 . m_1 = h_1 :: t_1 \Rightarrow m_2 = h_1 :: t_1 \Rightarrow (\mathbf{r}_1\ h_1\ t_1) \sqsubseteq (\mathbf{r}_2\ h_2\ t_2)$

where Φ is the inductive hypothesis obtained from the [LETREC] application. The first two premises follow directly from the definition of $\sqsubseteq$, while the third one follows from the contradictory assumptions $m_1 \sqsubseteq_g m_2$, $m_1 = h_1 :: t_1$ and $m_2 = []$. The last premise is proved by first applying the [$\mathsf{NATCASE\text{-}R}$] rule and then applying the induction hypothesis. $\square$

The proof presented here is intended to show how the one-sided rules can deal with asynchronous reasoning, but we remark that a much simpler proof could be written using equational rewriting rules. However, note that our system can also be seen as a framework in which to embed and prove sound such rewriting rules, in the style of Benton (2004).

8.3 Third example: Selection sort

This example showcases a property, namely sensitivity, that is out of reach of equational reasoning, but that is easy to prove using relational reasoning.

Given two lists of integers of the same length, we define the distance between them as the maximum of the pointwise distances:

$$\begin{array}{lll} d(l_1, l_2) & \triangleq & \max_i |l_1[i] - l_2[i]| \quad \text{if } l_1, l_2 \text{ have the same length} \\ d(l_1, l_2) & \triangleq & \infty \quad \text{otherwise} \end{array}$$

It is routine to check that this defines in fact a metric. Furthermore, it is known that sorting is 1-Lipschitz continuous under this metric: If $\mathsf{sort}(l)$ denotes the list obtained by sorting l, then, for all l_1, l_2 we have that $d(\mathsf{sort}(l_1), \mathsf{sort}(l_2)) \leq d(l_1, l_2)$.

This result can be proved directly by showing that for any k the function $\min_k$, which picks the $k-$th smallest element of a list, is 1-Lipschitz continuous. The above result then follows by noting that $\mathsf{sort}(l) = [min_1(l), \min_2(l), ..., \min_n(l)]$ where n is the length of l.

Here, we provide a different proof. We prove that a particular implementation of sorting, namely, selection sort has this property. Selection sort is a basic sorting algorithm that traverses a list, finds the least element, puts it in front, and then sorts the rest of the list recursively. The function ssort below implements selection sort.

```
$\mathsf{ssort} l \triangleq \mathsf{ssort}'\ l\ (\mathsf{length}\ l)$

$\mathsf{ssort}' \triangleq$ letrec $\mathsf{ssort}'\ l =$
        $\lambda n.$case $n$ of
                $0 \quad \mapsto [];$
                $S \quad \mapsto \lambda m.$case $l$ of
                    $[] \ \mapsto []$
                    $\_ :: \_ \ \mapsto \lambda ht.$    let$(rest, min) = \mathsf{restmin}\ t\ h$
                                        in $min :: \mathsf{ssort}'\ rest\ m$

$\mathsf{restmin} \triangleq$ letrec $restmin\ l =$
        $\lambda a.$case $l$ of
                    $[] \mapsto ([], a)$
                    $\_ :: \_ \mapsto \lambda ht.$let   $M = \max(a, h)$
                                        $m = \min(a, h)$
                                        $(rest, min) = \mathsf{restmin}\ m\ t$
                                        in $\langle M :: rest, min \rangle$
```

We then want to prove the following in RHOL:

Theorem 24 (*1-Lipschitz continuity of ssort*)

$$\vdash \mathsf{ssort} \sim \mathsf{ssort} \mid \forall l_1 l_2 \delta. |l_1| = |l_2| \Rightarrow d(l_1, l_2) \leq \delta \Rightarrow d(\mathbf{r}_1\ l_1, \mathbf{r}_2\ l_2) \leq \delta$$

Note that the postcondition above is equivalent to

$$\forall l_1 l_2. |l_1| = |l_2| \Rightarrow d(\mathbf{r}_1\ l_1, \mathbf{r}_2\ l_2) \leq d(l_1, l_2)$$

but our version is easier to prove, because we need an explicit δ to appear in the verification of ssort' and restmin. The proof is entirely synchronous, and relies on the property of restmin that, for two lists of equal length satisfying the invariant of being closer than δ under the metric defined above, (1) the two returned minimas are spaced less than δ (under the usual metric for the integers) and (2) the two returned remaining lists satisfy the same invariant. In RHOL, this is expressed as:

$$\vdash \mathsf{restmin} \sim \mathsf{restmin} \mid \forall l_1 l_2. d(l_1, l_2) \leq \delta \Rightarrow \forall h_1 h_2. |h_1 - h_2| \leq \delta \Rightarrow D(\mathbf{r}_1\ l_1\ h_1, \mathbf{r}_2\ l_2\ h_2) \leq \delta$$

where we use D to denote the distance induced by the maximum of the component-wise distances.

The only interesting case is where we reach the $h :: t$ branch of $\mathsf{restmin}$. Here, we use the following (mathematical) lemma about max and min. This lemma must be proved in HOL with sufficient axiomatization.

Lemma 25
Let δ a non-negative real number. For every a_1, b_1, a_2, b_2 real, if $|a_1 - a_2| \leq \delta$ and $|b_1 - b_2| \leq \delta$, then:

$$|\max(a_1, b_1) - \max(a_2, b_2)| \leq \delta \text{ and}$$
$$|\min(a_1, b_1) - \min(a_2, b_2)| \leq \delta$$

8.4 Fourth Example: Insertion Sort

Insertion sort is a standard sorting algorithm that sorts a list $h :: t$ by sorting the tail t recursively and then inserting h at the appropriate position in the sorted tail. Consider the following implementations of the insertion function, insert, and the insertion sort function, isort, each returning a pair, whose first element is the usual output list (inserted list for insert and sorted list for isort) and whose second element is the *number of comparisons* made during the execution (assuming an eager, call-by-value evaluation strategy).

```
$\mathsf{insert} \triangleq \lambda x.\,\text{letrec } insert\ l = \text{case } l \text{ of } [] \mapsto ([x], 0);$
                                   $\_ :: \_ \mapsto \lambda h t.\,\text{case } x \leq h \text{ of}$
                                                   $\mathsf{tt} \mapsto (x :: l, 1);$
                                                   $\mathsf{ff} \mapsto \text{let } s = insert\ t \text{ in}$
                                                         $(h :: (\pi_1\ s), 1 + (\pi_2\ s))$
$\mathsf{isort} \triangleq \text{letrec } isort\ l = \text{case } l \text{ of } [] \mapsto ([], 0);$
                              $\_ :: \_ \mapsto \lambda h t.\ \text{let } s = isort\ t$
                                            $\text{let } s' = \mathsf{insert}\ h\ (\pi_1\ s) \text{ in}$
                                            $(\pi_1\ s', (\pi_2\ s) + (\pi_2\ s'))$
```

Using this implementation, we prove the following interesting fact about insertion sort: Among all lists of the same length, insertion sort computes the fastest (with fewest comparisons) on lists that are already sorted. This is a property about the relational cost of insertion sort (on two different inputs), which cannot be established in RelCost. To state the property in RHOL, we define a list predicate $\mathsf{sorted}(l)$ in HOL axiomatically:

$$\mathsf{sorted}([]) \equiv \top \qquad\qquad \forall h t.\,\mathsf{sorted}(h :: t) \equiv (\mathsf{sorted}(t) \wedge h \leq \mathsf{lmin}(t))$$

where the function $\mathsf{lmin}(l)$ returns the minimum element of l:

$$\mathsf{lmin} \triangleq \text{letrec } f\ l = \text{case } l \text{ of } [] \mapsto \infty; \_ :: \_ \mapsto \lambda h t.\,\mathsf{min}(h, f\ t)$$

As in the previous example, let $|\cdot|$ be the standard list length function. The property of insertion sort mentioned above is formalized in the following theorem. In words, the theorem says that if isort is executed on lists x_1 and x_2 of the same length and

x_1 is sorted, then the number of comparisons made during the sorting of x_1 is no more than the number of comparisons made during the sorting of x_2.

Theorem 26
Let $\tau \triangleq \mathsf{list}_{\mathbb{N}} \to \mathsf{list}_{\mathbb{N}}$. Then, $\bullet \mid \bullet \vdash \mathsf{isort} : \tau \sim \mathsf{isort} : \tau \mid \forall x_1\, x_2. (\mathsf{sorted}(x_1) \wedge |x_1| = |x_2|) \Rightarrow \pi_2(\mathbf{r}_1\ x_1) \leq \pi_2(\mathbf{r}_2\ x_2)$.

A full proof is shown in the appendix (Section D.5). The proof proceeds mostly synchronously in the two sides. Following the structure of isort, we apply the rules [LETREC], [LISTCASE] and [APP] + [ABS] (for the let binding, which, as usual, is defined as a function application), followed by an application of the inductive hypothesis for the recursive call to *isort*. Eventually, we expose the call to insert on both sides. At this point, the observation is that since x_1 is already sorted, its head element must be no greater than all elements in its tail, so insert must return immediately with at most 1 comparison on the x_1 side. Formally, this last proof step can be completed by switching to either UHOL or HOL and using subject conversion; in the appendix, we switch to HOL.

9 Implementation

We have mechanized our system in the Coq proof assistant (The Coq Development Team, 2018). Instead of building the mechanization of the system over a set-theoretic model, we build it over the Calculus of Inductive Constructions that underlies Coq via a shallow embedding. In other words, we make a slight change to the system—in place of PCF (Section 2) and HOL (Section 4) as the language and the underlying logic of refinements, we use Coq's language and Coq's `Prop` logic, respectively. In addition to allowing us to leverage Coq directly for proofs in the underlying logic, this also shows that our syntax-directed unary and relational rules are not particularly tied to set-theory or HOL. Another reason for using Coq is that it supports type quantification, which we exploit in our embedding (see the definition of j_rhol below). We believe that a mechanization could also be carried out in a HOL-based proof assistant like Isabelle/HOL but since these assistants typically lack support for type quantification, the mechanization would be more involved (Wildmoser & Nipkow, 2004).

A judgment in RHOL (similarly for UHOL) is mechanized as a function that receives an element of type `A1`, an element of type `A2` and a function of type `A1 -> A2 -> Prop` and returns a `Prop`, namely the result of applying the latter to the former.

```
Definition j_rhol : forall (A1 A2: Type),
  A1 -> A2 -> (A1 -> A2 -> Prop) -> Prop :=
  fun A1 A2 t1 t2 P => P t1 t2.

Notation "|- t1 ; A1 ~ t2 ; A2 | P" := (j_rhol A1 A2 t1 t2 P)
```

Our relational and unary proof rules are mechanized as lemmas. We describe as an example the application rule. We require that the arguments `t1` and `t2` be related by P, and that the functions f1 and f2 send arguments related by P to results related by Q. This can be described through the following lemma.

```
Lemma App2 :
  forall (A1 A2 B1 B2 : Type)
         (P : A1 -> A2 -> Prop) (Q: A1 -> A2 -> B1 -> B2 -> Prop)
         (f1 : A1 -> B1) (f2 : A2 -> B2) (t1 : A1) (t2 : A2),
     (|- t1 ; A1 ~ t2 ; A2 | P) ->
     (|- f1 ; A1 -> B1 ~ f2 ; A2 -> B2 |
         (fun r1 r2 => forall (x1: A1) (x2 : A2),
              P x1 x2 -> Q x1 x2 (r1 x1) (r2 x2))) ->
     (|- (f1 t1) ; B1 ~ (f2 t2) ; B2 | Q t1 t2).
```

Working with a shallow embedding allows us to use Coq directly to manage most of the type and logical context. So, in general, they do not need to explicitly appear in the rules. However, notice that we need to pass t1 and t2 as arguments to Q since they appear in its context. Compare this to the way the quantified variables get replaced by the arguments in the conclusion of the [APP] rule in Section 6.

The choice of Coq as a base language also allows us to have a more general recursion rule, that inducts on some arbitrary well-founded order:

```
Lemma RecWF2 :
  forall (A B1 B2 : Type)
         (P : A -> A -> Prop) (Q : A -> A -> B1 -> B2 -> Prop)
         (f1 : A -> B1) (f2 : A -> B2)
         (R : A -> A -> Prop),
     (well_founded R) ->
     (forall (x1 x2 : A),
         P x1 x2 -> (forall y1 y2,
                       R y1 x1 -> R y2 x2 -> P y1 y2 ->
                         Q y1 y2 (f1 y1) (f2 y2)) ->
         |- (f1 x1) ; B1 ~ (f2 x2) ; B2 | Q x1 x2)  ->
      |- f1 ; (A -> B1) ~ f2 ; (A -> B2) |
         (fun r1 r2 => forall (x1 x2: A),
            P x1 x2 -> Q x1 x2 (r1 x1) (r2 x2)).
```

Since the embedding is shallow, the proof of a lemma is also a proof of soundness of the rule that the lemma is implementing. The choice of a shallow embedding makes the proofs go through smoothly, and most of them can be fully automated with the default auto tactic. This takes care of proving the forward implications (soundness) in Theorems 3 and 6. To prove the reverse implications (relative completeness) we need additional theorems, which can also be proven with auto. For instance, the relative completeness of RHOL is expressed as

```
Theorem rhol_complete : forall (A1 : Type) (A2 : Type)
                               (t1 : A1) (t2 : A2)
                               (P : A1 -> A2 -> Prop),
    (wt_uhol A1 t1) -> (wt_uhol A2 t2) -> P t1 t2 ->
    |- t1 ; A1 ~ t2 ; A2 | P.
Proof.
 auto.
Qed.
```

where `wt_uhol` is a predicate expressing well-typedness.

Our implementation also includes the proof of the two examples presented in Sections 8.1 and Section 8.2. We verified these using the rules as lemmas. See for example the following (abridged) proof of the first example.

```
Theorem fact_equiv :
  |- factorial ; nat -> nat ~ factorial_acc ; nat -> nat -> nat |
     fun r1 r2 => forall n1 n2, n1 = n2 ->
        forall k, ((r1 n1)*k)%nat = r2 n2 k.
Proof.
  (** We start by applying the 2-sided recursion rule *)
  apply (RecNat2 _ _ _
          (fun _ _ r1 r2 =>
            forall k, (r1*k)%nat = r2 k) factorial factorial_acc).
  intros x1 x2 Heq IH.
  auto_absR.
  (** We unfold the definitions *)
  (* 2 lines omitted *)
  (** We start the case analysis *)
  apply CaseNat2.
  destruct Heq.
  - (** We first need to show that both terms
        take the same branch.*)
    easy.
  - (** Now we prove the 0 ~ 0 case *)
    auto with arith.
  - (** Finally we show the S ~ S case. *)
    auto_abs2.
    apply Var2.
    (** The rest of the proof is basic reasoning in
        FOL with Arithmetic. We just need to
        instantiate the induction hypothesis IH  *)
    (* 10 lines omitted *)
    ring.
Qed.
```

Here, the `auto_abs` tactics provide automation for some common patterns of application of the rule [ABS]. Notice also how the structure of the Coq proof follows the proof in Section 8.1.

10 Extensions

The system presented in this paper allows us to prove relational properties about pure programs. Our system has four ingredients:

1. A base language and its typing rules (PCF)
2. A logic over such programs, based on inference rules (HOL)
3. A system of refinements over the type system defined in the first step (UHOL), where the refinements are expressed in the logic of the second step
4. A similar system of refinements on pair of programs (RHOL), which uses the unary system defined in the third step in the one-sided rules

The way these components interact is key to having a system that allows for an informal style of reasoning while retaining completeness with respect to the base logic. Abstracting a bit, the four steps above can be seen as a general recipe for building a syntax-directed system for proving relational properties of programs. The key idea is to keep types and refinements separate but, at the same time, make the refinements mirror the construction of the types in such a way that type and logical inference can be done simultaneously (see, e.g., how the logical implication in the refinement of function abstraction mirrors the arrow in the type). Finding the right way to achieve this is crucial in building unary and relational systems from the base language's type system and the base logic.

The work presented in this paper only considers simply-typed terms, but systems for reasoning about pure programs are already very common in the literature. A question that may arise is whether the approach used in this paper could also be used to build a relational logic for a more expressive language and base logic, while retaining an informal reasoning style and relative completeness. The answer to this question is affirmative: since the publication of the conference version of this article (Aguirre *et al.*, 2017), a few systems based on the ideas of RHOL have been developed to reason about programs in richer languages with *effects*. While the individual designs of these systems are guided by the effects considered and by the relational properties of interest, they all use recipes similar to the one described above. The common challenge in the design of these systems is identifying abstraction mechanisms which permit reasoning about the effects and their relational properties in a natural way. These abstraction mechanisms often also require changes to the underlying logic. We comment on some of these systems briefly.

Monadic Relational Cost (Radicek *et al.*, 2018). This work starts from a PCF-like language that has a monad to track the cost of a computation. The base logic is HOL extended with principles to establish equality of monadic computations with costs. The combination of functional refinements with the cost monad results in a very expressive system where the proof of the cost of a computation may depend on functional properties. Following the recipe described here, the work develops two syntax-directed systems: a unary system U^{C} to reason about the cost of a single computation, and a relational system R^{C} to reason about the difference in the costs of two computations.

Guarded RHOL (Aguirre *et al.*, 2018). This work extends the present article in two ways: (1) it adds the *later* modality to the language and the base logic, which allows reasoning about infinite data structures inductively, and (2) it adds a monad of discrete probability distributions. Combined, the two extensions allow the representation of, and reasoning about, stochastic processes such as Markov Chains. The relational system syntax-directed system uses *probabilistic couplings*, a common tool from probability theory, to express relations between pairs of Markov Chains. One-sided rules help in proving properties of unsynchronized runs of a pair of Markov Chains.

Probabilistic RHOL (Sato *et al.*, 2019). This work presents a relational logic to reason about PCF_p, a higher-order language with probabilistic sampling and Bayesian conditioning. It presents a new base logic, PL, to express and prove properties of probabilistic programs in PCF_p. On top of this logic, two syntax-directed systems are built using the recipe described above—a unary one (UPL) and a relational one (RPL).

11 Conclusion

We have developed Relational Higher-Order Logic (RHOL), a new formalism to reason about relational properties of (pure) higher-order programs written in a simply typed λ-calculus with inductive types and recursive definitions. The system is expressive, has solid foundations via an equivalence with Higher-Order Logic, and yet retains the (nice) "feel" of relational refinement type systems. An important direction for future work is to extend Relational Higher-Order Logic to other kinds of effects, in particular mutable state.

For practical purposes, it will also be interesting to automate RHOL. We believe that much of the technology developed for (relational) refinement types, and in particular the automated generation of verification conditions (maybe with user hints to switch between unary and binary modes of reasoning) and the connection to SMT-solvers can be lifted without significant hurdle to Relational Higher-Order Logic.

References

Abadi, Martín, Cardelli, Luca, & Curien, Pierre-Louis. (1993). Formal parametric polymorphism. *Pages 157–170 of: Conference Record of the Twentieth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Charleston, South Carolina, USA, January 1993.*

Abadi, Martín, Banerjee, Anindya, Heintze, Nevin, & Riecke, Jon G. (1999). A core calculus of dependency. *Pages 147–160 of: POPL '99, Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Antonio, TX, USA, January 20-22, 1999.*

Aczel, Peter, & Gambino, Nicola. (2000). Collection principles in dependent type theory. *Pages 1–23 of:* Callaghan, Paul, Luo, Zhaohui, McKinna, James, & Pollack, Robert (eds), *Types for Proofs and Programs, International Workshop, TYPES 2000, Durham, UK, December 8-12, 2000, Selected Papers.* Lecture Notes in Computer Science, vol. 2277. Springer.

Aczel, Peter, & Gambino, Nicola. (2006). The generalised type-theoretic interpretation of constructive set theory. *J. symb. log.*, **71**(1), 67–103.

Adams, Robin, & Luo, Zhaohui. (2010). Classical predicative logic-enriched type theories. *Ann. pure appl. logic*, **161**(11), 1315–1345.

Aguirre, Alejandro, Barthe, Gilles, Gaboardi, Marco, Garg, Deepak, & Strub, Pierre-Yves. (2017). A relational logic for higher-order programs. *PACMPL*, **1**(ICFP), 21:1–21:29.

Aguirre, Alejandro, Barthe, Gilles, Birkedal, Lars, Bizjak, Ales, Gaboardi, Marco, & Garg, Deepak. (2018). Relational reasoning for markov chains in a probabilistic guarded lambda calculus. *Pages 214–241 of:* Ahmed, Amal (ed), *Programming Languages and Systems -*

27th European Symposium on Programming, ESOP 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings. Lecture Notes in Computer Science, vol. 10801. Springer.

Alpern, Bowen, & Schneider, Fred B. (1985). Defining liveness. *Inf. process. lett.*, **21**(4), 181–185.

Asada, Kazuyuki, Sato, Ryosuke, & Kobayashi, Naoki. (2016). Verifying relational properties of functional programs by first-order refinement. *Science of computer programming.*

Barthe, Gilles, D'Argenio, Pedro R., & Rezk, Tamara. (2004). Secure information flow by self-composition. *Pages 100–114 of: 17th IEEE Computer Security Foundations Workshop, (CSFW-17 2004), 28-30 June 2004, Pacific Grove, CA, USA.*

Barthe, Gilles, Grégoire, Benjamin, & Béguelin, Santiago Zanella. (2009). Formal certification of code-based cryptographic proofs. *Pages 90–101 of: Proceedings of the 36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2009, Savannah, GA, USA, January 21-23, 2009.*

Barthe, Gilles, Crespo, Juan Manuel, & Kunz, César. (2011). Relational verification using product programs. *Pages 200–214 of: FM 2011: Formal Methods - 17th International Symposium on Formal Methods, Limerick, Ireland, June 20-24, 2011. Proceedings.*

Barthe, Gilles, Köpf, Boris, Olmedo, Federico, & Béguelin, Santiago Zanella. (2012). Probabilistic relational reasoning for differential privacy. *Pages 97–110 of: Proceedings of the 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2012, Philadelphia, Pennsylvania, USA, January 22-28, 2012.*

Barthe, Gilles, Fournet, Cédric, Grégoire, Benjamin, Strub, Pierre-Yves, Swamy, Nikhil, & Béguelin, Santiago Zanella. (2014). Probabilistic relational verification for cryptographic implementations. *Pages 193–206 of:* Jagannathan, Suresh, & Sewell, Peter (eds), *Proceedings of the 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL'14.*

Barthe, Gilles, Gaboardi, Marco, Gallego Arias, Emilio Jesús, Hsu, Justin, Roth, Aaron, & Strub, Pierre-Yves. (2015). Higher-order approximate relational refinement types for mechanism design and differential privacy. *Pages 55–68 of:* Rajamani, Sriram K., & Walker, David (eds), *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015.*

Barthe, Gilles, Grégoire, Benjamin, Hsu, Justin, & Strub, Pierre-Yves. (2017). Coupling proofs are probabilistic product programs. *Pages 161–174 of: Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017.*

Belo, João Filipe. (2007). Dependently sorted logic. *Pages 33–50 of:* Miculan, Marino, Scagnetto, Ivan, & Honsell, Furio (eds), *Types for Proofs and Programs, International Conference, TYPES 2007, Cividale del Friuli, Italy, May 2-5, 2007, Revised Selected Papers*. Lecture Notes in Computer Science, vol. 4941. Springer.

Benton, Nick. (2004). Simple relational correctness proofs for static analyses and program transformations. *Pages 14–25 of:* Jones, Neil D., & Leroy, Xavier (eds), *Proceedings of the 31th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL'04.*

Beringer, Lennart, & Hofmann, Martin. (2007). Secure information flow and program logics. *Pages 233–248 of: 20th IEEE Computer Security Foundations Symposium, CSF 2007, 6-8 July 2007, Venice, Italy*. IEEE Computer Society.

Blatter, Lionel, Kosmatov, Nikolai, Gall, Pascale Le, & Prevosto, Virgile. (2017). Deductive verification with relational properties. *In Proc. of the 23th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2017), Uppsala, Sweden.* To Appear.

Çiçek, Ezgi, Barthe, Gilles, Gaboardi, Marco, Garg, Deepak, & Hoffmann, Jan. (2017). Relational cost analysis. *Pages 316–329 of:* Castagna, Giuseppe, & Gordon, Andrew D. (eds), *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017.* ACM.

Clarkson, Michael R., & Schneider, Fred B. (2008). Hyperproperties. *Pages 51–65 of: Proceedings of CSF'08.*

Dreyer, Derek, Neis, Georg, Rossberg, Andreas, & Birkedal, Lars. (2010). A relational modal logic for higher-order stateful adts. *Pages 185–198 of: Proceedings of the 37th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2010, Madrid, Spain, January 17-23, 2010.*

Dreyer, Derek, Ahmed, Amal, & Birkedal, Lars. (2011). Logical step-indexed logical relations. *Logical methods in computer science*, **7**(2).

Dunfield, Joshua, & Pfenning, Frank. (2004). Tridirectional typechecking. *Pages 281–292 of:* Jones, Neil D., & Leroy, Xavier (eds), *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2004, Venice, Italy, January 14-16, 2004.* ACM.

Dybjer, Peter. (1985). Program verification in a logical theory of constructions. *Pages 334–349 of:* Jouannaud, Jean-Pierre (ed), *Functional Programming Languages and Computer Architecture, FPCA 1985, Nancy, France, September 16-19, 1985, Proceedings.* Lecture Notes in Computer Science, vol. 201. Springer.

Freeman, Timothy S., & Pfenning, Frank. (1991). Refinement types for ML. *Pages 268–277 of:* Wise, David S. (ed), *Proceedings of the ACM SIGPLAN'91 Conference on Programming Language Design and Implementation (PLDI), Toronto, Ontario, Canada, June 26-28, 1991.* ACM.

Gaboardi, Marco, Haeberlen, Andreas, Hsu, Justin, Narayan, Arjun, & Pierce, Benjamin C. (2013). Linear dependent types for differential privacy. *Pages 357–370 of:* Giacobazzi, Roberto, & Cousot, Radhia (eds), *The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '13, Rome, Italy - January 23 - 25, 2013.* ACM.

Ghani, Neil, Forsberg, Fredrik Nordvall, & Simpson, Alex. (2016a). Comprehensive parametric polymorphism: Categorical models and type theory. *Pages 3–19 of: Foundations of Software Science and Computation Structures - 19th International Conference, FOSSACS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings.*

Ghani, Neil, Forsberg, Fredrik Nordvall, & Simpson, Alex. (2016b). Comprehensive parametric polymorphism: Categorical models and type theory. *Pages 3–19 of:* Jacobs, Bart, & Löding, Christof (eds), *Foundations of Software Science and Computation Structures - 19th International Conference, FOSSACS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings.* Lecture Notes in Computer Science, vol. 9634. Springer.

Grimm, Niklas, Maillard, Kenji, Fournet, Cédric, Hritcu, Catalin, Maffei, Matteo, Protzenko, Jonathan, Ramananandro, Tahina, Rastogi, Aseem, Swamy, Nikhil, & Béguelin, Santiago Zanella. (2018). A monadic framework for relational verification: applied to information security, program equivalence, and optimizations. *Pages 130–145 of:* Andronick, June, & Felty, Amy P. (eds), *Proceedings of the 7th ACM SIGPLAN*

International Conference on Certified Programs and Proofs, CPP 2018, Los Angeles, CA, USA, January 8-9, 2018. ACM.

Hatcliff, John, & Danvy, Olivier. (1997). A computational formalization for partial evaluation. *Mathematical structures in computer science*, **7**, 507–541.

Heintze, Nevin, & Riecke, Jon G. (1998). The slam calculus: Programming with secrecy and integrity. *Pages 365–377 of: POPL '98, Proceedings of the 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Diego, CA, USA, January 19-21, 1998.*

Jacobs, B. (1999). *Categorical logic and type theory.* Studies in Logic and the Foundations of Mathematics, no. 141. Amsterdam: North Holland.

Jung, Ralf, Swasey, David, Sieczkowski, Filip, Svendsen, Kasper, Turon, Aaron, Birkedal, Lars, & Dreyer, Derek. (2015). Iris: Monoids and invariants as an orthogonal basis for concurrent reasoning. *Pages 637–650 of: Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015.*

Kobayashi, Naoki, Lozes, Étienne, & Bruse, Florian. (2017). On the relationship between higher-order recursion schemes and higher-order fixpoint logic. *Pages 246–259 of: Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017.*

Kobayashi, Naoki, Tsukada, Takeshi, & Watanabe, Keiichi. (2018). Higher-order program verification via HFL model checking. *Pages 711–738 of: Programming Languages and Systems - 27th European Symposium on Programming, ESOP 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings.*

Krogh-Jespersen, Morten, Svendsen, Kasper, & Birkedal, Lars. (2017). A relational model of types-and-effects in higher-order concurrent separation logic. *Pages 218–231 of: Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017.*

Melliès, Paul-André, & Zeilberger, Noam. (2015). Functors are type refinement systems. *Pages 3–16 of:* Rajamani, Sriram K., & Walker, David (eds), *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015.* ACM.

Nanevski, Aleksandar, Banerjee, Anindya, & Garg, Deepak. (2013). Dependent type theory for verification of information flow and access control policies. *ACM trans. program. lang. syst.*, **35**(2), 6:1–6:41.

Pfenning, Frank. (2008). Church and Curry: Combining intrinsic and extrinsic typing. *Pages 303–338 of:* C.Benzmüller, C.Brown, J.Siekmann, & R.Statman (eds), *Reasoning in Simple Type Theory: Festschrift in Honor of Peter B. Andrews on His 70th Birthday.* Studies in Logic 17. College Publications.

Plotkin, Gordon. (1973). *Lambda-definability and logical relations.*

Plotkin, Gordon. (1977). Lcf considered as a programming language. *Theoretical computer science*, **5**(3), 223 – 255.

Plotkin, Gordon D., & Abadi, Martín. (1993). A logic for parametric polymorphism. *Pages 361–375 of: Typed Lambda Calculi and Applications, International Conference on Typed Lambda Calculi and Applications, TLCA '93, Utrecht, The Netherlands, March 16-18, 1993, Proceedings.*

Pottier, François, & Simonet, Vincent. (2002). Information flow inference for ML. *Pages 319–330 of: Conference Record of POPL 2002: The 29th SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Portland, OR, USA, January 16-18, 2002.*

Radicek, Ivan, Barthe, Gilles, Gaboardi, Marco, Garg, Deepak, & Zuleger, Florian. (2018). Monadic refinements for relational cost analysis. *PACMPL*, **2**(POPL), 36:1–36:32.

Sato, Tetsuya, Aguirre, Alejandro, Barthe, Gilles, Gaboardi, Marco, Garg, Deepak, & Hsu, Justin. (2019). Formal verification of higher-order probabilistic programs: reasoning about approximation, convergence, bayesian inference, and optimization. *PACMPL*, **3**(POPL), 38:1–38:30.

Sousa, Marcelo, & Dillig, Isil. (2016). Cartesian hoare logic for verifying k-safety properties. *Pages 57–69 of: Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2016, Santa Barbara, CA, USA, June 13-17, 2016.*

Statman, R. (1985). Logical relations and the typed λ-calculus. *Information and control*, **65**(2-3), 85–97.

Stewart, Gordon, Banerjee, Anindya, & Nanevski, Aleksandar. (2013). Dependent types for enforcement of information flow and erasure policies in heterogeneous data structures. *Pages 145–156 of: 15th International Symposium on Principles and Practice of Declarative Programming, PPDP '13, Madrid, Spain, September 16-18, 2013.*

Swamy, Nikhil, Hritcu, Catalin, Keller, Chantal, Rastogi, Aseem, Delignat-Lavaud, Antoine, Forest, Simon, Bhargavan, Karthikeyan, Fournet, Cédric, Strub, Pierre-Yves, Kohlweiss, Markulf, Zinzindohoue, Jean Karim, & Béguelin, Santiago Zanella. (2016). Dependent types and multi-monadic effects in F. *Pages 256–270 of:* Bodík, Rastislav, & Majumdar, Rupak (eds), *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016.* ACM.

Tait, William W. (1967). Intensional interpretations of functionals of finite type I. *J. symb. log.*, **32**(2), 198–212.

Terauchi, Tachio, & Aiken, Alex. (2005). Secure information flow as a safety problem. *Pages 352–367 of:* Hankin, Chris, & Siveroni, Igor (eds), *Static Analysis Symposium.* lncs, vol. 3672.

The Coq Development Team. 2018 (Apr.). *The coq proof assistant, version 8.8.0.*

Unno, Hiroshi, Torii, Sho, & Sakamoto, Hiroki. (2017). Automating induction for solving horn clauses. *Pages 571–591 of: Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II.*

Vazou, Niki, Seidel, Eric L., Jhala, Ranjit, Vytiniotis, Dimitrios, & Jones, Simon L. Peyton. (2014). Refinement types for haskell. *Pages 269–282 of:* Jeuring, Johan, & Chakravarty, Manuel M. T. (eds), *Proceedings of the 19th ACM SIGPLAN international conference on Functional programming, Gothenburg, Sweden, September 1-3, 2014.* ACM.

Volpano, Dennis, Smith, Geoffrey, & Irvine, Cynthia. (1996). A sound type system for secure flow analysis. *Journal of computer security*, **4**(3), 1–21.

Wildmoser, Martin, & Nipkow, Tobias. (2004). Certifying machine code safety: Shallow versus deep embedding. *Pages 305–320 of: Theorem Proving in Higher Order Logics, 17th International Conference, TPHOLs 2004, Park City, Utah, USA, September 14-17, 2004, Proceedings.*

Xi, Hongwei, & Pfenning, Frank. (1999). Dependent types in practical programming. *Pages 214–227 of:* Appel, Andrew W., & Aiken, Alex (eds), *POPL '99, Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Antonio, TX, USA, January 20-22, 1999.* ACM.

Yang, Hongseok. (2007). Relational separation logic. *Theoretical computer science*, **375**(1-3), 308–334.

Zaks, Anna, & Pnueli, Amir. (2008). CoVaC: Compiler Validation by Program Analysis of the Cross-Product. *Pages 35–51 of:* Cuéllar, Jorge, Maibaum, T. S. E., & Sere, Kaisa (eds), *Formal Methods*. Lecture Notes in Computer Science, vol. 5014.

Zeilberger, Noam. (2016). *Principles of type refinement*. Notes for OPLSS 2016 school.

A Semantics

A.1 Semantics of HOL

A.1.1 Types

The interpretation for the types corresponds directly to the usual representation of pairs, lists and functions in set theory.

$$\begin{aligned}
[\![\mathbb{B}]\!] &\triangleq \{\mathsf{ff}, \mathsf{tt}\}\\
[\![\mathbb{N}]\!] &\triangleq \mathbb{N}\\
[\![\mathsf{list}_\tau]\!] &\triangleq \mathsf{list}_{[\![\tau]\!]}\\
[\![\tau_1 \times \tau_2]\!] &\triangleq [\![\tau_1]\!] \times [\![\tau_2]\!]\\
[\![\tau_1 \to \tau_2]\!] &\triangleq [\![\tau_1]\!] \to [\![\tau_2]\!]
\end{aligned}$$

A.1.2 Terms

The terms are given an interpretation with respect to a valuation ρ which is a partial function mapping variables to elements in the interpretation of their type. Given ρ, we use the notation $\rho[v/x]$ to denote the unique extension of ρ such that if $y = x$ then $\rho[v/x](y) = v$ and, otherwise, $\rho[v/x](y) = \rho(y)$.

$$(\!|x|\!)_\rho \triangleq \rho(x) \qquad (\!|\langle t, u\rangle|\!)_\rho := \langle (\!|t|\!)_\rho, (\!|u|\!)_\rho\rangle \qquad (\!|\pi_i\ t|\!)_\rho \triangleq \pi_i((\!|t|\!)_\rho)$$

$$(\!|\lambda x : \tau.t|\!)_\rho \triangleq \lambda v : [\![\tau]\!].(\!|x|\!)_{\rho[(\!|v|\!)_\rho/v]} \qquad (\!|c|\!)_\rho \triangleq c \qquad (\!|S\ t|\!)_\rho \triangleq S\ (\!|t|\!)_\rho$$

$$(\!|t :: u|\!)_\rho \triangleq (\!|t|\!)_\rho :: (\!|u|\!)_\rho$$

$$(\!|\text{case } t \text{ of } [] \mapsto u; \_ :: \_ \mapsto v|\!)_\rho \triangleq \begin{cases} (\!|u|\!)_\rho & \text{if } (\!|t|\!)_\rho = [] \\ (\!|v|\!)_\rho\ M\ N & \text{if } (\!|t|\!)_\rho = M :: N \end{cases}$$

$$(\!|\text{letrec } f\ x = t|\!)_\rho \triangleq F \ \text{ where } F \text{ is the unique solution of the fixpoint equation}$$

A.1.3 Formulas

We assume that for predicate P of arity $\tau_1 \times \cdots \times \tau_n$, we have an interpretation $[\![P]\!] \in [\![\tau_1]\!] \times \cdots \times [\![\tau_n]\!]$ that satisfies the axioms for P. The interpretation of a formula is defined as follows:

$$\begin{array}{rcl}
(\!|P(t_1,\ldots,t_n)|\!)_\rho & \triangleq & ([\![t_1]\!]_\rho,\ldots,[\![t_n]\!]_\rho) \in [\![P]\!] \\
(\!|\top|\!)_\rho & \triangleq & \tilde{\top} \\
(\!|\bot|\!)_\rho & \triangleq & \tilde{\bot} \\
(\!|\phi_1 \wedge \phi_2|\!)_\rho & \triangleq & (\!|\phi_1|\!)_\rho \mathrel{\tilde{\wedge}} (\!|\phi_2|\!)_\rho \\
(\!|\phi_1 \Rightarrow \phi_2|\!)_\rho & \triangleq & (\!|\phi_1|\!)_\rho \mathrel{\tilde{\Rightarrow}} (\!|\phi_2|\!)_\rho \\
(\!|\forall x:\tau.\phi|\!)_\rho & \triangleq & \tilde{\forall} v. v \in [\![\tau]\!] \mathrel{\tilde{\Rightarrow}} (\!|\phi|\!)_{\rho[v/x]}
\end{array}$$

where we use the tilde ($\sim$) to distinguish between the (R)HOL connectives and the meta-level connectives.

A.1.4 Soundness

We have the following result:

Theorem 27 (*Soundness of set-theoretical semantics*)
If $\Gamma \mid \Psi \vdash \phi$, then for every valuation $\rho \models \Gamma$, $\bigwedge_{\psi\in\Psi} (\!|\psi|\!)_\rho$ implies $(\!|\phi|\!)_\rho$.

Proof
By induction on the length of the derivation of $\Gamma \mid \Psi \vdash \phi$. □

A.2 Semantics of UHOL

The intended meaning of a UHOL judgment $\Gamma \mid \Psi \vdash t : \tau \mid \phi$ is:

$$\text{for all } \rho.\ \text{s.t. } \rho \models \Gamma,\ (\!|\bigwedge \Psi|\!)_\rho \ \text{ implies } \ (\!|\phi|\!)_{\rho[(\!|t|\!)_\rho/\mathbf{r}]}$$

We have the following result:

Theorem 28 (*Set-theoretical soundness and consistency of UHOL*)
If $\Gamma \mid \Psi \vdash t : \sigma \mid \phi$, then for every valuation $\rho \models \Gamma$, $\bigwedge_{\psi\in\Psi} (\!|\psi|\!)_\rho$ implies $(\!|\phi|\!)_{\rho[(\!|t|\!)_\rho/\mathbf{r}]}$. In particular, there is no proof of $\Gamma \mid \emptyset \vdash t : \sigma \mid \bot$ in UHOL.

Proof
It is a direct consequence of the embedding from UHOL into HOL and the soundness of HOL. □

A.3 Semantics of RHOL

The intended meaning of a RHOL judgment $\Gamma \mid \Psi \vdash t_1 : \tau_1 \sim t_2 : \tau_2 \mid \phi$ is:

$$\text{for all } \rho.\ \text{s.t. } \rho \models \Gamma,\ (\!|\bigwedge \Psi|\!)_\rho \ \text{ implies } \ (\!|\phi|\!)_{\rho[(\!|t_1|\!)_\rho/\mathbf{r}_1][(\!|t_2|\!)_\rho/\mathbf{r}_2]}$$

We have the following result:

Theorem 29 (*Set-theoretical soundness and consistency of RHOL*)

If $\Gamma \mid \Psi \vdash t_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi$, then for every valuation $\rho \models \Gamma$, $\bigwedge_{\psi\in\Psi} (\!|\psi|\!)_\rho$ implies $(\!|\phi|\!)_{\rho[(\!|t_1|\!)_\rho/\mathbf{r}_1],[(\!|t_2|\!)_\rho/\mathbf{r}_2]}$. In particular, there is no proof of $\Gamma \mid \emptyset \vdash t_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \bot$ for any Γ.

Proof

It is a direct consequence of the embedding of RHOL into HOL and the soundness of HOL. □

B Additional rules

For reasons of space, we have omitted some derivable and admissible rules in HOL, UHOL and RHOL. These are useful to prove some theorems and examples. We now discuss the most interesting among them:

B.1 HOL

The following rules are derivable in HOL:

- A cut rule can be derived from $[\Rightarrow_I]$ and $[\Rightarrow_E]$:

$$\frac{\Gamma \mid \Psi, \phi' \vdash \phi \quad \Gamma \mid \Psi \vdash \phi'}{\Gamma \mid \Psi \vdash \phi}\ \textsf{CUT}$$

- A rule for case analysis can be derived from [LIST]:

$$\frac{\Gamma \vdash l : \mathsf{list}_\tau \quad \Gamma \mid \Psi, l = [] \vdash \phi \quad \Gamma, h : \tau, t : \mathsf{list}_\tau \mid \Psi, l = h :: t \vdash \phi}{\Gamma \mid \Psi \vdash \phi}\ \textsf{DESTR}-\textsf{LIST}$$

- A rule [S-LIST] for strong induction can be derived from [LIST]:

$$\frac{\Gamma \mid \Psi \vdash \phi[[]/t] \quad \Gamma, h : \tau, t : \mathsf{list}_\tau \mid \Psi, \forall u : \mathsf{list}_\tau . |u| \le |t| \Rightarrow \phi[u/t] \vdash \phi[h :: t/t]}{\Gamma \mid \Psi \vdash \forall t : \mathsf{list}_\tau . \phi}$$

- A rule [D-LIST] for (weak) double induction can be derived by applying [LIST] twice:

$$\frac{\begin{array}{c}\Gamma \mid \Psi \vdash \phi[[]/l_1][[]/l_2] \\ \Gamma, h_1 : \tau_1, t_1 : \mathsf{list}_{\tau_1} \mid \Psi, \phi[t_1/l_1][[]/l_2] \vdash \phi[h_1 :: t_1/l_1][[]/l_2] \\ \Gamma, h_2 : \tau_2, t_2 : \mathsf{list}_{\tau_2} \mid \Psi, \phi[[]/l_1][t_2/l_2] \vdash \phi[[]/l_1][h_2 :: t_2/l_2] \\ \Gamma, h_1 : \tau_1, t_2 : \mathsf{list}_{\tau_2}, h_2 : \tau_2, t_2 : \mathsf{list}_{\tau_2} \mid \Psi, \phi[t_1/l_1][t_2/l_2] \vdash \phi[h_1 :: t_1/l_1][h_2 :: t_2/l_2]\end{array}}{\Gamma \mid \Psi \vdash \forall l_1 l_2 . \phi}$$

- A rule [S-D-LIST] for strong double induction can be derived from [D-LIST]:

$$\frac{\begin{array}{c}\Gamma \mid \Psi \vdash \phi[[]/l_1][[]/l_2] \\ \Gamma, h_1 : \tau_1, t_1 : \mathsf{list}_{\tau_1} \mid \Psi, \forall m_1. |m_1| \leq |t_1| \Rightarrow \phi[m_1/l_1][[]/l_2] \vdash \phi[h_1 :: t_1/l_1][[]/l_2] \\ \Gamma, h_2 : \tau_2, t_2 : \mathsf{list}_{\tau_2} \mid \Psi, \forall m_2. |m_2| \leq |t_2| \Rightarrow \phi[[]/l_1][m_2/l_2] \vdash \phi[[]/l_1][h_2 :: t_2/l_2] \\ \Gamma, h_1 : \tau_1, t_1 : \mathsf{list}_{\tau_1}, h_2 : \tau_2, t_2 : \mathsf{list}_{\tau_2} \mid \\ \Psi, \forall m_1 m_2. (|m_1|, |m_2|) < (|h_1 :: t_1|, |h_2 :: t_2|) \Rightarrow \phi[m_1/l_1][m_2/l_2] \vdash \\ \phi[h_1 :: t_1/l_1][h_2 :: t_2/l_2]\end{array}}{\Gamma \mid \Psi \vdash \forall l_1 l_2. \phi}$$

B.2 RHOL

The following version [NATCASE*] of the case rule with an extra premise on the case guards is admissible:

$$\frac{\begin{array}{c}\Gamma \mid \Psi \vdash t_1 : \mathsf{list}_{\tau_1} \sim t_2 : \mathsf{list}_{\tau_2} \mid \phi' \wedge (\mathbf{r}_1 = 0 \Leftrightarrow \mathbf{r}_2 = 0) \\ \Gamma \mid \Psi, \phi'[0/\mathbf{r}_1][0/\mathbf{r}_2] \vdash u_1 : \sigma_1 \sim u_2 : \sigma_2 \mid \phi \\ \Gamma \mid \Psi \vdash v_1 : \mathbb{N} \to \sigma_1 \sim v_2 : \mathbb{N} \to \sigma_2 \mid \forall x_1 x_2. \phi'[Sx_1/\mathbf{r}_1][Sx_2/\mathbf{r}_2] \Rightarrow \phi[\mathbf{r}_1\ x_1/\mathbf{r}_1][\mathbf{r}_2\ x_2/\mathbf{r}_2]\end{array}}{\Gamma \mid \Psi \vdash \mathrm{case}\ t_1\ \mathsf{of}\ 0 \mapsto u_1; S \mapsto v_1 : \sigma_1 \sim \mathrm{case}\ t_2\ \mathsf{of}\ 0 \mapsto u_2; S \mapsto v_2 : \sigma_2 \mid \phi}$$

The one sided version is admissible as well:

$$\frac{\begin{array}{c}\Gamma \mid \Psi \vdash t_1 : \mathsf{list}_{\tau_1} \sim t_2 : \sigma_2 \mid \phi' \\ \Gamma \mid \Psi, \phi'[0/\mathbf{r}_1][t_2/\mathbf{r}_2] \vdash u_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi \\ \Gamma \mid \Psi \vdash v_1 : \mathbb{N} \to \sigma_1 \sim t_2 : \sigma_2 \mid \forall x_1. \phi'[Sx_1/\mathbf{r}_1] \Rightarrow \phi[\mathbf{r}_1\ x_1/\mathbf{r}_1]\end{array}}{\Gamma \mid \Psi \vdash \mathrm{case}\ t_1\ \mathsf{of}\ 0 \mapsto u_1; S \mapsto v_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi}\ \mathsf{NATCASE}*{-}\mathsf{L}$$

Notice that we can always recover the initial version of the rule by instantiating ϕ' as $t_1 = \mathbf{r}_1 \wedge t_2 = \mathbf{r}_2$.

C Proofs

C.1 Proof of Theorem 6

The easier direction is the reverse implication. To prove it, one just notices that we can trivially apply [SUB] instantiating ϕ' as a tautology that matches the structure of the types, For instance, for a base type $\mathbb{N}$ we would use $\top$, for an arrow type $\mathbb{N} \to \mathbb{N}$ we would use $\forall x. \bot \Rightarrow \top$, and so on.

We now prove the direct implication by induction on the derivation of $\Gamma \mid \Psi \vdash t_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi$. Suppose the last rule is:
Case. [VAR] (similarly, [NIL] and [PROJ])
The premise of the rule is already the judgment we want to prove.

Case [ABS]. The rule is

$$\frac{\Gamma, x_1 : \tau_1, x_2 : \tau_2 \mid \Psi, \phi' \vdash t_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi}{\Gamma \mid \Psi \vdash \lambda x_1.t_1 : \tau_1 \to \sigma_1 \sim \lambda x_2.t_2 : \tau_2 \to \sigma_2 \mid \forall x_1, x_2. \phi' \Rightarrow \phi[\mathbf{r}_1\ x_1/\mathbf{r}_1][\mathbf{r}_2\ x_2/\mathbf{r}_2]}$$

By applying the induction hypothesis on the premise:
$\Gamma, x_1 : \tau_1, x_2 : \tau_2 \mid \Psi, \phi' \vdash \phi[t_1/\mathbf{r}_1][t_2/\mathbf{r}_2]$ (1)
By applying $[\Rightarrow_I]$ on (1):
$\Gamma, x_1 : \tau_1, x_2 : \tau_2 \mid \Psi \vdash \phi' \Rightarrow \phi[t_1/\mathbf{r}_1][t_2/\mathbf{r}_2]$ (2)
By applying $[\forall_I]$ twice on (2):
$\Gamma \mid \Psi \vdash \forall x_1 x_2 . \phi' \Rightarrow \phi[t_1/\mathbf{r}_1][t_2/\mathbf{r}_2]$ (3)
Finally, by applying CONV on (3):
$\Gamma \mid \Psi \vdash \forall x_1 x_2 . \phi' \Rightarrow \phi[(\lambda x_1 . t_1)\ x_1/\mathbf{r}_1][(\lambda x_2 . t_2)\ x_2/\mathbf{r}_2]$
Proof for [ABS-L] (and [ABS-R]) is analogous.

Case [APP]. The rule is

$$\frac{\begin{array}{c}\Gamma \mid \Psi \vdash t_1 : \tau_1 \to \sigma_1 \sim t_2 : \tau_2 \to \sigma_2 \mid \forall x_1, x_2 . \phi'[x_1/\mathbf{r}_1][x_2/\mathbf{r}_2] \Rightarrow \phi[\mathbf{r}_1\ x_1/\mathbf{r}_1][\mathbf{r}_2\ x_2/\mathbf{r}_2] \\ \Gamma \mid \Psi \vdash u_1 : \tau_1 \sim u_2 : \tau_2 \mid \phi'\end{array}}{\Gamma \mid \Psi \vdash t_1 u_1 : \sigma_1 \sim t_2 u_2 : \sigma_2 \mid \phi[u_1/x_1][u_2/x_2]}$$

By applying the induction hypothesis on the premises we have:
$\Gamma \mid \Psi \vdash \forall x_1 x_2 . \phi'[x_1/\mathbf{r}_1][x_2/\mathbf{r}_2] \Rightarrow \phi[t_1\ x_1/\mathbf{r}_1][t_2\ x_2/\mathbf{r}_2]$ (1)
and
$\Gamma \mid \Psi \vdash \phi'[u_1/\mathbf{r}_1][u_2/\mathbf{r}_2]$ (2)
By applying twice $[\forall_E]$ to (1) with u_1, u_2:
$\Gamma \mid \Psi \vdash \phi'[u_1/\mathbf{r}_1][u_2/\mathbf{r}_2] \Rightarrow \phi[t_1\ u_1/\mathbf{r}_1][t_2\ u_2/\mathbf{r}_2]$ (3)
and by applying $[\Rightarrow_E]$ to (3) and (2):
$\Gamma \mid \Psi \vdash \phi[t_1\ u_1/\mathbf{r}_1][t_2\ u_2/\mathbf{r}_2]$
Proof for [APP-L] (and APP-R) is analogous, and it uses the UHOL embedding for the premise about the argument. Proofs for [CONS] and [PAIR] are very similar as well.

Case [SUB]. The rule is

$$\frac{\Gamma \mid \Psi \vdash t_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi' \qquad \Gamma \mid \Psi \vdash_{\mathsf{HOL}} \phi'[t_1/\mathbf{r}_1][t_2/\mathbf{r}_2] \Rightarrow \phi[t_1/\mathbf{r}_1][t_2/\mathbf{r}_2]}{\Gamma \mid \Psi \vdash t_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi}$$

Applying the inductive hypothesis on the premises we have:
$\Gamma \mid \Psi \vdash \phi'[t_1/\mathbf{r}_1][t_2/\mathbf{r}_2]$
and
$\Gamma \mid \Psi \vdash \phi'[t_1/\mathbf{r}_1][t_2/\mathbf{r}_2] \Rightarrow \phi[t_1/\mathbf{r}_1][t_2/\mathbf{r}_2]$
The proof is simply applying $[\Rightarrow_E]$.

Case [LETREC]. The rule is

$$\frac{\begin{array}{c}\mathcal{D}ef(f_1,x_1,e_1)\ \mathcal{D}ef(f_2,x_2,e_2)\\ \Gamma, x_1 : I_1, x_2 : I_2, f_1 : I_1 \to \sigma, f_2 : I_2 \to \sigma_2 \mid \Psi, \phi',\\ \forall m_1 m_2.(|m_1|,|m_2|) < (|x_1|,|x_2|) \Rightarrow \phi'[m_1/x_1][m_2/x_2] \Rightarrow\\ \phi[m_1/x_1][m_2/x_2][f_1\ m_1/\mathbf{r}_1][f_2\ m_2/\mathbf{r}_2] \vdash\\ e_1 : \sigma_1 \sim e_2 : \sigma_2 \mid \phi\end{array}}{\begin{array}{c}\Gamma \mid \Psi \vdash \text{letrec}\ f_1\ x_1\ = e_1 : I_1 \to \sigma_2 \sim \text{letrec}\ f_2\ x_2\ = e_2 : I_2 \to \sigma_2 \mid\\ \forall x_1 x_2.\phi' \Rightarrow \phi[\mathbf{r}_1\ x_1/\mathbf{r}_1][\mathbf{r}_2\ x_2/\mathbf{r}_2]\end{array}}$$

As an example, we prove the list and nat case, but for other datatypes the proof is similar. Applying the inductive hypothesis on the premise we have:

$$\begin{array}{c}\Gamma, l_1, n_2, f_1, f_2 \mid \Psi, \forall m_1 m_2.(|m_1|,|m_2|) < (|l_1|,|n_2|) \Rightarrow \phi[f_1 m_1/\mathbf{r}_1][f_2 m_2/\mathbf{r}_2] \vdash\\ \phi[e_1/\mathbf{r}_1][e_2/\mathbf{r}_2]\end{array}$$

By $[\forall_I]$ we derive:

$$\begin{array}{c}\Gamma \mid \Psi \vdash \forall f_1, f_2, l_1, n_2.(\forall m_1 m_2.(|m_1|,|m_2|) < (|l_1|,|n_2|) \Rightarrow\\ \phi[f_1 m_1/\mathbf{r}_1][f_2 m_2/\mathbf{r}_2]) \Rightarrow \phi[e_1/\mathbf{r}_1][e_2/\mathbf{r}_2]\end{array}. \qquad (\Phi)$$

We want to prove

$$\Gamma \mid \Psi \vdash \forall l_1 n_2.\phi[F_1\ l_1/\mathbf{r}_1][F_2\ n_2/\mathbf{r}_2]$$

where we use the abbreviations

$$\begin{array}{rcl}F_1 & := & \text{letrec}\ f_1\ x_1 = e_1\\ F_2 & := & \text{letrec}\ f_2\ x_2 = e_2\end{array}$$

We will use strong double induction over natural numbers and lists. We need to prove four premises. Since we can prove (Φ) from Γ, Ψ, we can add it to the axioms:

(A) $\Gamma \mid \Psi, \Phi \vdash \phi[F_1\ []/\mathbf{r}_1][F_2\ 0/\mathbf{r}_2]$
(B) $\Gamma, h_1, t_1 \mid \Psi, \Phi, \forall m_1.|m_1| \le |t_1| \Rightarrow \phi[F_1\ m_1/\mathbf{r}_1][F_2\ 0/\mathbf{r}_2] \vdash \phi[F_1\ (h_1 :: t_1)/\mathbf{r}_1][F_2\ 0/\mathbf{r}_2]$
(C) $\Gamma, x_2 \mid \Psi, \Phi, \forall m_2.|m_2| \le |x_2| \Rightarrow \phi[F_1\ []/\mathbf{r}_1][F_2\ m_2/\mathbf{r}_2] \vdash \phi[F_1\ []/\mathbf{r}_1][F_2\ (Sx_2)/\mathbf{r}_2]$
(D) $\Gamma, h_1, t_1, x_2 \mid \Psi, \Phi, \forall m_1 m_2.(|m_1|,|m_2|) < (|h_1 :: t_1|, |Sx_2|) \Rightarrow$
$\phi[F_1\ m_1/\mathbf{r}_1][F_2\ m_2/\mathbf{r}_2] \vdash \phi[F_1\ (h_1 :: t_1)/\mathbf{r}_1][F_2\ (Sx_2)/\mathbf{r}_2]$

To prove them, we will instantiate the quantifiers in Φ with the appropriate variables.

To prove (A), we instantiate Φ at $F_1, F_2, [], 0$:

$$\begin{array}{c}(\forall m_1 m_2.(|m_1|,|m_2|) < (|[]|,|0|) \Rightarrow \phi[F_1 m_1/\mathbf{r}_1][F_2 m_2/\mathbf{r}_2]) \Rightarrow\\ \phi[e_1/\mathbf{r}_1][e_2/\mathbf{r}_2][[]/l_1][0/n_2][F_1/f_1][F_2/f_2]\end{array}$$

and, since $(|m_1|,|m_2|) < (|[]|,|0|)$ is trivially false, then

$$\phi[e_1/\mathbf{r}_1][e_2/\mathbf{r}_2][[]/l_1][0/n_2][F_1/f_1][F_2/f_2]$$

and by beta-expansion and [CONV]:

$$\phi[F_1\ []/\mathbf{r}_1][F_2\ 0/\mathbf{r}_2]$$

.

To prove (B), we instantiate Φ at $F_1, F_2, h_1 :: t_1, 0$

$$(\forall m_1 m_2.(|m_1|,|m_2|) < (|h_1 :: t_1|,|0|) \Rightarrow \phi[F_1 m_1/\mathbf{r}_1][F_2 m_2/\mathbf{r}_2]) \Rightarrow \\ \phi[e_1/\mathbf{r}_1][e_2/\mathbf{r}_2][h_1 :: t_1/l_1][0/n_2][F_1/f_1][F_2/f_2]$$

by beta-expansion:

$$(\forall m_1 m_2.(|m_1|,|m_2|) < (|h_1 :: t_1|,|0|) \Rightarrow \phi[F_1 m_1/\mathbf{r}_1][F_2 m_2/\mathbf{r}_2]) \Rightarrow \\ \phi[F_1\ h_1 :: t_1/\mathbf{r}_1][F_2\ 0/\mathbf{r}_2]$$

Since $(|m_1|,|m_2|) < (|h_1 :: t_1|,|0|)$ is only satisfied if $|m_1| \leq |t_1| \wedge m_2 = 0$, we can write it as:

$$(\forall m_1 m_2.(|m_1| \leq |t_1| \wedge m_2 = 0) \Rightarrow \phi[F_1 m_1/\mathbf{r}_1][F_2 m_2/\mathbf{r}_2]) \Rightarrow \phi[F_1\ h_1 :: t_1/\mathbf{r}_1][F_2\ 0/\mathbf{r}_2]$$

Meanwhile, one of the antecedents of (B) is $\forall m_1.|m_1| \leq |t_1| \Rightarrow \phi[F_1\ m_1/\mathbf{r}_1][F_2\ 0/\mathbf{r}_2]$, so by $[\Rightarrow_E]$ we prove $\phi[F_1\ h_1 :: t_1/\mathbf{r}_1][F_2\ 0/\mathbf{r}_2]$, which is the consequent of (B).

The proof of (C) is symmetrical to the proof of (B).

To prove (D), we instantiate Φ at $F_1, F_2, h_1 :: t_1, Sx_2$

$$(\forall m_1 m_2.(|m_1|,|m_2|) < (|h_1 :: t_1|,|Sx_2|) \Rightarrow \phi[F_1 m_1/\mathbf{r}_1][F_2 m_2/\mathbf{r}_2]) \Rightarrow \\ \phi[e_1/\mathbf{r}_1][e_2/\mathbf{r}_2][h_1 :: t_1/l_1][Sx_2/n_2][F_1/f_1][F_2/f_2]$$

by beta-expansion:

$$(\forall m_1 m_2.(|m_1|,|m_2|) < (|h_1 :: t_1|,|Sx_2|) \Rightarrow \phi[F_1 m_1/\mathbf{r}_1][F_2 m_2/\mathbf{r}_2]) \Rightarrow \\ \phi[F_1\ h_1 :: t_1/\mathbf{r}_1][F_2\ (Sx_2)/\mathbf{r}_2]$$

One of the antecedents of (D) is exactly $\forall m_1 m_2.(|m_1|,|m_2|) < (|h_1 :: t_1|,|Sx_2|) \Rightarrow \phi[F_1\ m_1/\mathbf{r}_1][F_2\ m_2/\mathbf{r}_2]$, so by $[\Rightarrow_E]$ we prove $\phi[F_1\ h_1 :: t_1/\mathbf{r}_1][F_2\ (Sx_2)/\mathbf{r}_2]$, which is the consequent of (D).

Proof of [LETREC-L] (and [LETREC-R]) is analogous, and uses simple strong induction.

Case [CASE]. The rule:

$$\frac{\begin{array}{c}\Gamma \mid \Psi \vdash l_1 : \mathsf{list}_{\tau_1} \sim l_2 : \mathsf{list}_{\tau_2} \mid \mathbf{r}_1 = [] \Leftrightarrow \mathbf{r}_2 = [] \\ \Gamma \mid \Psi, l_1 = [], l_2 = [] \vdash u_1 : \sigma_1 \sim u_2 : \sigma_2 \mid \phi \\ \Gamma \mid \Psi \vdash v_1 : \tau_1 \to \mathsf{list}_{\tau_1} \to \sigma_1 \sim v_2 : \tau_2 \to \mathsf{list}_{\tau_2} \to \sigma_2 \mid \\ \forall h_1 h_2 t_1 t_2. l_1 = h_1 :: t_1 \Rightarrow l_2 = h_2 :: t_2 \Rightarrow \phi[\mathbf{r}_1\ h_1\ t_1/\mathbf{r}_1][\mathbf{r}_2\ h_2\ t_2/\mathbf{r}_2]\end{array}}{\Gamma \mid \Psi \vdash \text{case } l_1 \text{ of } [] \mapsto u_1; \_ :: \_ \mapsto v_1 : \sigma_1 \sim \text{case } l_2 \text{ of } [] \mapsto u_2; \_ :: \_ \mapsto v_2 : \sigma_2 \mid \phi}$$

We prove the rule for natural numbers. Applying the induction hypothesis to the premises of the rule, we have:

(A) $\Gamma \mid \Psi \vdash t_1 = 0 \Leftrightarrow t_2 = 0$
(B) $\Gamma \mid \Psi, t_1 = 0, t_2 = 0 \vdash \phi[u_1/\mathbf{r}_1][u_2/\mathbf{r}_2]$
(C) $\Gamma \mid \Psi \vdash \forall x_1, x_2. t_1 = Sx_1 \Rightarrow t_2 = Sx_2 \Rightarrow \phi[v_1\ x_1/\mathbf{r}_1][v_2\ x_2/\mathbf{r}_2]$

We want to prove:

$$\Gamma \mid \Psi \vdash \phi[(\text{case } t_1 \text{ of } 0 \mapsto u_1; S \mapsto v_1)/\mathbf{r}_1][(\text{case } t_2 \text{ of } 0 \mapsto u_2; S \mapsto v_2)/\mathbf{r}_2]$$

By applying [DESTR-NAT] twice, we get four premises:

1. $\Gamma \mid \Psi, t_1 = 0, t_2 = 0 \vdash \phi[(\text{case } t_1 \text{ of } 0 \mapsto u_1; S \mapsto v_1)/\mathbf{r}_1][(\text{case } t_2 \text{ of } 0 \mapsto u_2; S \mapsto v_2)/\mathbf{r}_2]$
2. $\Gamma, m_2 \mid \Psi, t_1 = 0, t_2 = Sm_2 \vdash \phi[(\text{case } t_1 \text{ of } 0 \mapsto u_1; S \mapsto v_1)/\mathbf{r}_1][(\text{case } t_2 \text{ of } 0 \mapsto u_2; S \mapsto v_2)/\mathbf{r}_2]$
3. $\Gamma, m_1 \mid \Psi, t_1 = Sm_1, t_2 = 0 \vdash \phi[(\text{case } t_1 \text{ of } 0 \mapsto u_1; S \mapsto v_1)/\mathbf{r}_1][(\text{case } t_2 \text{ of } 0 \mapsto u_2; S \mapsto v_2)/\mathbf{r}_2]$
4. $\Gamma, m_1, m_2 \mid \Psi, t_1 = Sm_1, t_2 = Sm_2 \vdash \phi[(\text{case } t_1 \text{ of } 0 \mapsto u_1; S \mapsto v_1)/\mathbf{r}_1][(\text{case } t_2 \text{ of } 0 \mapsto u_2; S \mapsto v_2)/\mathbf{r}_2]$

We can prove (2) and (3) by deriving a contradiction with [NC] and (A). After beta-reducing in (1) and (4) we can easily derive them from (B) and (C) respectively.

Proof of [CASE-L] (and [CASE-R]) is analogous.

C.2 Proof of Lemma 10

By the embedding into HOL, we have:

- $\Gamma \mid \Psi \vdash \phi[t_1/\mathbf{r}]$
- $\Gamma \mid \Psi \vdash \phi'[t_2/\mathbf{r}]$

and by the $[\wedge_I]$ rule,

$$\Gamma \mid \Psi \vdash \phi[t_1/\mathbf{r}] \wedge \phi'[t_2/\mathbf{r}].$$

Finally, by undoing the embedding:

$$\Gamma \mid \Psi \vdash t_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi.$$

C.3 Proof of Theorem 11

By induction on the derivation:

Case. $x : \tau, \Gamma \vdash x : \tau$
To prove : $x : |\tau|, |\Gamma| \vdash \lfloor \tau \rfloor(x), \lfloor \Gamma \rfloor \vdash x : |\tau| \mid \lfloor \tau \rfloor(\mathbf{r})$. Directly by [VAR].

Case. $\dfrac{\Gamma, x : \tau \vdash t : \sigma}{\Gamma \vdash \lambda x.t : \Pi(x : \tau).\sigma}$
To prove: $|\Gamma| \mid \lfloor \Gamma \rfloor \vdash \lambda x.t : |\Pi(x : \tau).\sigma| \mid \lfloor \Pi(x : \tau).\sigma \rfloor(\mathbf{r})$.
Expanding the definitions:
$|\Gamma| \mid \lfloor \Gamma \rfloor \vdash \lambda x.t : |\tau| \to |\sigma| \mid \forall x. \lfloor \tau \rfloor(x) \Rightarrow \lfloor \sigma \rfloor(\mathbf{r}x)$
By induction hypothesis on the premise:
$|\Gamma|, x : |\tau| \mid \lfloor \Gamma \rfloor, \lfloor \tau \rfloor(x) \vdash t : |\sigma| \mid \lfloor \sigma \rfloor(\mathbf{r})$
Directly by [ABS].

Case. $\dfrac{\Gamma \vdash t : \Pi(x : \tau).\sigma \qquad \Gamma \vdash u : \tau}{\Gamma \vdash t\ u : \sigma[u/x]}$
To prove: $|\Gamma| \mid \lfloor \Gamma \rfloor \vdash t\ u : |\sigma[u/x]| \mid \lfloor \sigma[u/x] \rfloor(\mathbf{r})$.

Expanding the definitions:
$|\Gamma| \mid \lfloor\Gamma\rfloor \vdash t\ e_2 : |\sigma| \mid \lfloor\sigma\rfloor(\mathbf{r})[u/x]$
By induction hypothesis on the premise:
$|\Gamma| \mid \lfloor\Gamma\rfloor \vdash t : |\tau| \to |\sigma| \mid \forall x.\lfloor\tau\rfloor(x) \Rightarrow \lfloor\sigma\rfloor(\mathbf{r}x)$
and
$|\Gamma| \mid \lfloor\Gamma\rfloor \vdash u : |\tau| \mid \lfloor\tau\rfloor(\mathbf{r})$
We get the result directly by [APP].

Case. $\dfrac{\Gamma \vdash t : \mathsf{list}_\tau \qquad \Gamma \vdash u : \sigma \qquad \Gamma \vdash v : \tau \to \mathsf{list}_\tau \to \sigma}{\Gamma \vdash \text{case } t \text{ of } [] \mapsto u; \_ :: \_ \mapsto v : \sigma}$
To prove: $|\Gamma| \mid \lfloor\Gamma\rfloor \vdash \text{case } t \text{ of } [] \mapsto u; \_ :: \_ \mapsto v : |\sigma| \mid \lfloor\sigma\rfloor(\mathbf{r})$
By induction hypothesis on the premises:
$|\Gamma| \mid \lfloor\Gamma\rfloor \vdash t : |\mathsf{list}_\tau| \mid \lfloor\mathsf{list}_\tau\rfloor(\mathbf{r}),$ (1)
$|\Gamma| \mid \lfloor\Gamma\rfloor \vdash u : |\sigma| \mid \lfloor\sigma\rfloor(\mathbf{r}),$ (2)
and
$|\Gamma| \mid \lfloor\Gamma\rfloor \vdash v : |\tau \to \mathsf{list}_\tau \to \sigma| \mid \lfloor\tau \to \mathsf{list}_\tau \to \sigma\rfloor(\mathbf{r})$ (3)
Expanding the definitions on (3) we get:
$|\Gamma| \mid \lfloor\Gamma\rfloor \vdash v : |\tau| \to |\mathsf{list}_\tau| \to |\sigma| \mid \forall x.\lfloor\tau\rfloor(x) \Rightarrow \forall y.\lfloor\mathsf{list}_\tau\rfloor(y) \Rightarrow \lfloor\sigma\rfloor(\mathbf{r}\ x\ y)$ (4)
And from (1), (2) and (4) we apply [$\mathsf{LISTCASE*}$] and we get the result. Notice that (2) and (4) are stronger than the premises of the rule, so we will first need to apply [SUB] to weaken them

Case. $\dfrac{\Gamma \vdash \tau}{\Gamma \vdash [] : \mathsf{list}_\tau}$
To prove: $|\Gamma| \mid \lfloor\Gamma\rfloor \vdash [] : |\mathsf{list}_\tau| \mid \lfloor\mathsf{list}_\tau\rfloor(\mathbf{r})$
Expanding the definitions: $|\Gamma| \mid \lfloor\Gamma\rfloor \vdash [] : \mathsf{list}_{|\tau|} \mid \mathrm{All}(\mathbf{r}, x, \lfloor\tau\rfloor(x))$
And by the definition of All for the empty case, trivially $\mathrm{All}([], x, \lfloor\tau\rfloor(x))$, so we apply the [$\mathsf{NIL}$] rule and we get the result.

Case. $\dfrac{\Gamma \vdash h : \tau \qquad \Gamma \vdash t : \mathsf{list}_\tau}{\Gamma \vdash h :: t : \mathsf{list}_\tau}$
To prove: $|\Gamma| \mid \lfloor\Gamma\rfloor \vdash h :: t : |\mathsf{list}_\tau| \mid \lfloor\mathsf{list}_\tau\rfloor(\mathbf{r})$.
Expanding the definitions: $|\Gamma| \mid \lfloor\Gamma\rfloor \vdash h :: t : \mathsf{list}_{|\tau|} \mid \mathrm{All}(\mathbf{r}, \lambda x.\lfloor\tau\rfloor(x))$.
By induction hypothesis on the premises, we have:
$|\Gamma| \mid \lfloor\Gamma\rfloor \vdash h : |\tau| \mid \lfloor\tau\rfloor(\mathbf{r})$
and
$|\Gamma| \mid \lfloor\Gamma\rfloor \vdash t : \mathsf{list}_{|\tau|} \mid \mathrm{All}(\mathbf{r}, \lambda x.\lfloor\tau\rfloor(x))$.
We complete the proof by the [CONS] rule and the definition of All in the inductive case.

Case. $\dfrac{\Gamma \vdash \tau \preceq \sigma \qquad \Gamma \vdash t : \tau}{\Gamma \vdash t : \sigma}$
To prove: $|\Gamma| \mid \lfloor\Gamma\rfloor \vdash t : |\sigma| \mid \lfloor\sigma\rfloor(\mathbf{r})$
and, since $|\sigma| \equiv |\tau|$, it is the same as writing
$|\Gamma| \mid \lfloor\Gamma\rfloor \vdash t : |\tau| \mid \lfloor\tau\rfloor(\mathbf{r})$

By induction hypothesis on the premises:
$|\Gamma|, x : |\tau| \mid \lfloor\Gamma\rfloor, \lfloor\tau\rfloor(x) \vdash \lfloor\sigma\rfloor(x)$
and
$|\Gamma| \mid \lfloor\Gamma\rfloor \vdash t : |\tau| \mid \lfloor\tau\rfloor(\mathbf{r})$
The proof is completed by applying $[\Rightarrow_I]$ to the first premise, and then [SUB].

Case. $\dfrac{\Gamma, x : \tau, f : \Pi(y : \{\mathbf{r} : \tau \mid y < x\}).\sigma[y/x] \vdash t : \sigma \qquad \mathcal{D}ef(f,x,t)}{\Gamma \vdash \text{letrec } f\ x = t : \Pi(x : \tau).\sigma}$
To prove: $|\Gamma| \mid \lfloor\Gamma\rfloor \vdash \text{letrec } f\ x = t : |\Pi(x : \tau).\sigma| \mid \lfloor\Pi(x : \tau).\sigma\rfloor(\mathbf{r})$
By induction hypothesis on the premise:
$|\Gamma|, x : |\tau|, f : |\tau| \to |\sigma| \mid \lfloor\Gamma\rfloor, \lfloor\tau\rfloor(x), \forall y. \lfloor\tau\rfloor(y) \wedge y < x \Rightarrow \lfloor\sigma[y/x]\rfloor(f y) \vdash t : |\sigma| \mid \lfloor\sigma\rfloor(\mathbf{r})$
Directly by [LETREC].

C.4 Proof of Theorem 12

We will use without proof the following results:

Lemma 30
If $\Gamma \vdash \tau \preceq \sigma$ in refinement types, then $|\tau| \equiv |\sigma|$.

Proof
By induction on the derivation. □

Lemma 31
For every type τ and expression e and variable $x \notin FV(\tau, e)$, $\lfloor\tau\rfloor(e) = \lfloor\tau\rfloor(x)[e/x]$

Proof
By structural induction. □

Now we proceed with the proof of the theorem.
By induction on the derivation:

Case. $\dfrac{\Gamma \vdash \tau}{\Gamma \vdash \tau \preceq \tau}$
To show: $|\Gamma|, x : |\tau| \mid \lfloor\tau\rfloor(x) \vdash \lfloor\tau\rfloor(x)$. Directly by [$\mathsf{AX}$].

Case. $\dfrac{\Gamma \vdash \tau_1 \preceq \tau_2 \qquad \Gamma \vdash \tau_2 \preceq \tau_3}{\Gamma \vdash \tau_1 \preceq \tau_3}$
To show: $|\Gamma|, x : |\tau_1| \mid \lfloor\Gamma\rfloor, \lfloor\tau_1\rfloor(x) \vdash \lfloor\tau_3\rfloor(x)$.
By induction hypothesis on the premises,
$|\Gamma|, x : |\tau_1| \mid \lfloor\Gamma\rfloor, \lfloor\tau_1\rfloor(x) \vdash \lfloor\tau_2\rfloor(x)$
and
$|\Gamma|, x : |\tau_2| \mid \lfloor\Gamma\rfloor, \lfloor\tau_2\rfloor(x) \vdash \lfloor\tau_3\rfloor(x)$.
We complete the proof by [CUT]. Notice that $|\tau_1| \equiv |\tau_2| \equiv |\tau_3|$.

Case. $\dfrac{\Gamma \vdash \tau_1 \preceq \tau_2}{\Gamma \vdash \mathsf{list}_{\tau_1} \preceq \mathsf{list}_{\tau_2}}$
To show: $|\Gamma|, x : |\mathsf{list}_{\tau_1}| \mid \lfloor\Gamma\rfloor, \lfloor\mathsf{list}_{\tau_1}\rfloor(x) \vdash \lfloor\mathsf{list}_{\tau_2}\rfloor(\mathbf{r})$

Expanding the definitions: $|\Gamma|, x : \mathsf{list}_{|\tau_1|} \mid \lfloor\Gamma\rfloor, \top \vdash \top$,
which is trivial.

Case. $\dfrac{\Gamma \vdash \{\mathbf{r} : \tau \mid \phi\}}{\Gamma \vdash \{\mathbf{r} : \tau \mid \phi\} \preceq \tau}$
To show: $|\Gamma|, x : |\{\mathbf{r} : \tau \mid \phi\}| \mid \lfloor\{\mathbf{r} : \tau \mid \phi\}\rfloor(x) \vdash \lfloor\tau\rfloor(x)$.
Expanding the definitions: $|\Gamma|, x : |\{\mathbf{r} : \tau \mid \phi\}| \mid \lfloor\tau\rfloor(x) \wedge \phi[x/\mathbf{r}] \vdash \lfloor\tau\rfloor(x)$
and now the proof is completed trivially by $[\wedge_E]$ and $[\mathsf{AX}]$.

Case. $\dfrac{\Gamma \vdash \tau \preceq \sigma \qquad \Gamma, \mathbf{r} : \tau \vdash \phi}{\Gamma \vdash \tau \preceq \{\mathbf{r} : \sigma \mid \phi\}}$
To show: $|\Gamma|, \mathbf{r} : |\tau| \vdash \lfloor\Gamma\rfloor, \lfloor\tau\rfloor(\mathbf{r}) \vdash \lfloor\{\mathbf{r} : \sigma \mid \phi\}\rfloor(\mathbf{r})$
Expanding the definition: $|\Gamma|, \mathbf{r} : |\tau| \mid \lfloor\Gamma\rfloor, \lfloor\tau\rfloor(\mathbf{r}) \vdash \lfloor\sigma\rfloor(\mathbf{r}) \wedge \phi$
By induction hypothesis on the premises we have:
$|\Gamma|, \mathbf{r} : |\tau| \mid \lfloor\Gamma\rfloor, \lfloor\tau\rfloor(\mathbf{r}) \vdash \lfloor\sigma\rfloor(\mathbf{r})$
and:
$|\Gamma|, \mathbf{r} : |\tau| \mid \lfloor\Gamma\rfloor, \lfloor\tau\rfloor(\mathbf{r}) \vdash \phi$
We complete the proof by applying the $[\wedge_I]$ rule.

Case. $\dfrac{\Gamma \vdash \sigma_2 \preceq \sigma_1 \qquad \Gamma, x : \sigma_2 \vdash \tau_1 \preceq \tau_2}{\Gamma \vdash \Pi(x : \sigma_1).\tau_1 \preceq \Pi(x : \sigma_2).\tau_2}$
To show: $|\Gamma|, f : |\Pi(x : \sigma_1).\tau_1| \mid \lfloor\Gamma\rfloor, \lfloor\Pi(x : \sigma_1).\tau_1\rfloor(f) \vdash \lfloor\Pi(x : \sigma_2).\tau_2\rfloor(f)$
Expanding the definitions:
$|\Gamma|, f : |\Pi(x : \sigma_1).\tau_1| \mid \lfloor\Gamma\rfloor, \forall x. \lfloor\sigma_1\rfloor(x) \Rightarrow \lfloor\tau_1\rfloor(fx) \vdash \forall x. \lfloor\sigma_2\rfloor(x) \Rightarrow \lfloor\tau_2\rfloor(fx)$
By the rules $[\forall_I]$ and $[\Rightarrow_I]$ it suffices to prove:
$|\Gamma|, f : |\Pi(x : \sigma_1).\tau_1|, x : |\sigma_2| \mid \lfloor\Gamma\rfloor, \forall x. \lfloor\sigma_1\rfloor(x) \Rightarrow \lfloor\tau_1\rfloor(fx), \lfloor\sigma_2\rfloor(x) \vdash \lfloor\tau_2\rfloor(fx)$ (1)
On the other hand, by induction hypothesis on the premises:
$|\Gamma|, x : |\sigma_2| \mid \lfloor\Gamma\rfloor, \lfloor\sigma_2\rfloor(x) \vdash \lfloor\sigma_1\rfloor(x)$ (2)
and
$|\Gamma|, x : |\sigma_2|, y : |\tau_1| \mid \lfloor\Gamma\rfloor, \lfloor\sigma_2\rfloor(x), \lfloor\tau_1\rfloor(y) \vdash \lfloor\tau_2\rfloor(y)$ (3)
which we can weaken respectively to:
$|\Gamma|, x : |\sigma_2|, f : |\Pi(x : \sigma_1).\tau_1| \mid |\Gamma|, \lfloor\sigma_2\rfloor(x), \forall x. \lfloor\sigma_1\rfloor(x) \Rightarrow \lfloor\tau_1\rfloor(fx) \vdash \lfloor\sigma_1\rfloor(x)$ (4)
and
$|\Gamma|, x : |\sigma_2|, y : |\tau_1|, f : |\Pi(x : \sigma_1).\tau_1| \mid |\Gamma|, \lfloor\sigma_2\rfloor(x), \lfloor\tau_1\rfloor(y), \forall x. \lfloor\sigma_1\rfloor(x) \Rightarrow \lfloor\tau_1\rfloor(fx) \vdash$
$\lfloor\tau_2\rfloor(y)$ (5)
From (4), by doing a cut with its own premise $\forall x. \lfloor\sigma_1\rfloor(x) \Rightarrow \lfloor\tau_1\rfloor(fx)$, we derive:
$|\Gamma|, x : |\sigma_2|, f : |\Pi(x : \sigma_1).\tau_1| \mid \lfloor\Gamma\rfloor, \lfloor\sigma_2\rfloor(x), \forall x. \lfloor\sigma_1\rfloor(x) \Rightarrow \lfloor\tau_1\rfloor(fx) \vdash \lfloor\tau_1\rfloor(fx)$ (6)
From (5), by $[\Rightarrow_I]$ and $[\forall_I]$ we can derive:
$|\Gamma|, x : |\sigma_2|, f : |\Pi(x : \sigma_1).\tau_1| \mid \lfloor\Gamma\rfloor, \lfloor\sigma_2\rfloor(x), \forall x. \lfloor\sigma_1\rfloor(x) \Rightarrow \lfloor\tau_1\rfloor(fx) \vdash \forall y. \lfloor\tau_1\rfloor(y) \Rightarrow$
$\lfloor\tau_2\rfloor(y)$
And by $[\forall_E]$

$$\begin{array}{c} |\Gamma|, x : |\sigma_2|, f : |\Pi(x : \sigma_1).\tau_1| \mid \lfloor\Gamma\rfloor, \lfloor\sigma_2\rfloor(x), \forall x. \lfloor\sigma_1\rfloor(x) \Rightarrow \lfloor\tau_1\rfloor(fx) \vdash \\ \lfloor\tau_1\rfloor(fx) \Rightarrow \lfloor\tau_2\rfloor(fx) \end{array} \tag{7}$$

Finally, from (6) and (7) by $[\Rightarrow_E]$ we get:
$|\Gamma|, x : |\sigma_2|, f : |\Pi(x:\sigma_1).\tau_1| \mid \lfloor\Gamma\rfloor, \lfloor\sigma_2\rfloor(x), \forall x. \lfloor\sigma_1\rfloor(x) \Rightarrow \lfloor\tau_1\rfloor(fx) \vdash \lfloor\tau_2\rfloor(fx)$
and by one last application of $[\Rightarrow_I]$ we get what we wanted to prove.

C.5 Proof of Theorem 14

We can recover the lemma from the unary case:

Lemma 32
For every type τ, expressions t_1, t_2 and variables $x_1, x_2 \notin FV(\tau, t_1, t_2)$,

$$\llbracket\tau\rrbracket(t_1, t_2) = \llbracket\tau\rrbracket(x_1, x_2)[t_1/x_1][t_2/x_2]$$

Most cases are very similar to the unary case, so we will only show the most interesting ones:

Case. $\dfrac{\Gamma \vdash T}{\Gamma \vdash [] \sim [] :: \mathsf{list}_T}$
To show: $|\Gamma| \mid \llbracket\Gamma\rrbracket \vdash [] : |\mathsf{list}_T| \sim [] : |\mathsf{list}_T| \mid \llbracket\mathsf{list}_T\rrbracket(\mathbf{r}_1, \mathbf{r}_2)$.
There are two options. If T is a unary type, we have to prove:
$|\Gamma| \mid \llbracket\Gamma\rrbracket \vdash [] : |\mathsf{list}_T| \sim [] : |\mathsf{list}_T| \mid \bigwedge_{i\in\{1,2\}} \mathrm{All}(\mathbf{r}_i, \lambda x. \lfloor\tau\rfloor(x))$
And by the definition of All we can directly prove:
$\emptyset \mid \emptyset \vdash \mathrm{All}([], \lambda x. \lfloor\tau\rfloor(x)) \wedge \mathrm{All}([], \lambda x. \lfloor\tau\rfloor(x))$
If T is a relational type, we have to prove:
$|\Gamma| \mid \llbracket\Gamma\rrbracket \vdash [] : |\mathsf{list}_T| \sim [] : |\mathsf{list}_T| \mid \mathrm{All2}(\mathbf{r}_1, \mathbf{r}_2, \lambda x_1. \lambda x_2. \llbracket T\rrbracket(x_1, x_2))$
And by the definition of All2 we can directly prove:
$\emptyset \mid \emptyset \vdash \mathrm{All2}([], [], \lambda x_1. \lambda x_2. \llbracket T\rrbracket(x_1, x_2))$

Case. $\dfrac{\Gamma \vdash h_1 \sim h_2 :: T \qquad \Gamma \vdash t_1 \sim t_2 :: \mathsf{list}_T}{\Gamma \vdash h_1 :: t_1 \sim h_2 :: t_2 :: \mathsf{list}_T}$
To show: $|\Gamma| \mid \llbracket\Gamma\rrbracket \vdash h_1 :: t_2 : |\mathsf{list}_T| \sim h_2 :: t_2 : |\mathsf{list}_T| \mid \mathsf{list}_T$.
There are two options. If T is a unary type, we have to prove:
$|\Gamma| \mid \llbracket\Gamma\rrbracket \vdash h_1 :: t_1 : |\mathsf{list}_T| \sim h_2 :: t_2 : |\mathsf{list}_T| \mid \bigwedge_{i\in\{1,2\}} \mathrm{All}(\mathbf{r}_i, \lambda x. \lfloor T\rfloor(x))$
By induction hypothesis we have:
$|\Gamma| \mid \llbracket\Gamma\rrbracket \vdash h_1 : |T| \sim h_2 :: t_2 : |T| \mid \bigwedge_{i\in\{1,2\}} \lfloor T\rfloor(\mathbf{r}_i)$
and
$|\Gamma| \mid \llbracket\Gamma\rrbracket \vdash t_1 : |\mathsf{list}_T| \sim t_2 : |\mathsf{list}_T| \mid \bigwedge_{i\in\{1,2\}} \mathrm{All}(\mathbf{r}_i, \lambda x. \lfloor T\rfloor(x))$
And by the definition of All we can directly prove:
$\bigwedge_{i\in\{1,2\}} \lfloor T\rfloor(h_i) \Rightarrow \bigwedge_{i\in\{1,2\}} \mathrm{All}(t_i, \lambda x. \lfloor T\rfloor(x)) \Rightarrow \bigwedge_{i\in\{1,2\}} \mathrm{All}(h_i :: t_i, \lambda x. \lfloor T\rfloor(x))$
So by the [CONS] rule, we prove the result. If T is a relational type, we have to prove:
$|\Gamma| \mid \llbracket\Gamma\rrbracket \vdash h_1 :: t_1 : |\mathsf{list}_T| \sim h_2 :: t_2 : |\mathsf{list}_T| \mid \mathrm{All2}(\mathbf{r}_1, \mathbf{r}_2, \lambda x_1. \lambda x_2. \llbracket T\rrbracket(x_1, x_2))$
By induction hypothesis we have:
$|\Gamma| \mid \llbracket\Gamma\rrbracket \vdash h_1 : |T| \sim h_2 :: t_2 : |T| \mid \llbracket T\rrbracket(\mathbf{r}_1, \mathbf{r}_2)$
and
$|\Gamma| \mid \llbracket\Gamma\rrbracket \vdash t_1 : |\mathsf{list}_T| \sim t_2 : |\mathsf{list}_T| \mid \mathrm{All2}(\mathbf{r}_1, \mathbf{r}_2, \lambda x_1. \lambda x_2. \llbracket T\rrbracket(x_1, x_2))$
And by the definition of All2 we can directly prove:

$\llbracket T \rrbracket(h_1,h_2) \Rightarrow \mathrm{All2}(t_1,t_2,\lambda x_1.\lambda x_2.\llbracket T \rrbracket(x_1,x_2)) \Rightarrow \mathrm{All}(h_1 :: t_1, h_1 :: h_2, \lambda x_1.\lambda x_2.\llbracket T \rrbracket(x_1,x_2))$
So by the [CONS] rule, we prove the result.

Case. $\dfrac{\begin{array}{c}\Gamma \vdash t_1 \sim t_2 :: \mathsf{list}_T \\ \Gamma \vdash t_1 = [] \Leftrightarrow t_2 = [] \qquad \Gamma \vdash u_1 \sim u_2 :: U \qquad \Gamma \vdash v_1 \sim v_2 :: \Pi(h :: T).\Pi(t :: \mathsf{list}_T).U\end{array}}{\Gamma \vdash \text{case } t_1 \text{ of } [] \mapsto u_1; \_ :: \_ \mapsto v_1 \sim \text{case } t_2 \text{ of } [] \mapsto u_2; \_ :: \_ \mapsto v_2 :: U}$
To show:
$|\Gamma| \mid \llbracket \Gamma \rrbracket \vdash \text{case } t_1 \text{ of } [] \mapsto u_1; \_ :: \_ \mapsto v_1 : |U| \sim \text{case } t_2 \text{ of } [] \mapsto u_2; \_ :: \_ \mapsto r_2 : |U| \mid \llbracket U \rrbracket(\mathbf{r}_1,\mathbf{r}_2)$
By induction hypothesis we have:
$|\Gamma| \mid \llbracket \Gamma \rrbracket \vdash t_1 = [] \Leftrightarrow t_2 = []$,
$|\Gamma| \mid \llbracket \Gamma \rrbracket \vdash u_1 : |U| \sim u_2 : |U| \mid \llbracket U \rrbracket(\mathbf{r}_1,\mathbf{r}_2)$
and
$|\Gamma| \mid \llbracket \Gamma \rrbracket \vdash v_1 : T \to \mathsf{list}_T \to U \sim v_2 : T \to \mathsf{list}_T \to U \mid \forall h_1 h_2. \llbracket T \rrbracket(h_1,h_2) \Rightarrow \forall t_1 t_2. \llbracket \mathsf{list}_T \rrbracket(t_1,t_2) \Rightarrow \llbracket U \rrbracket(\mathbf{r}_1 h_1 t_1,\ h_2 t_2 \mathbf{r}_2)$
By applying the [LISTCASE*] rule to the three premises we get the result.

Case. $\dfrac{\begin{array}{c}\Gamma, x :: T, f :: \Pi(y :: \{y :: T \mid (y_1,y_2) < (x_1,x_2)\}).U[y/x] \vdash t_1 \sim t_2 :: U \\ \Gamma \vdash \Pi(x :: T).U \qquad \mathcal{D}ef(f_1,x_1,t_1) \qquad \mathcal{D}ef(f_2,x_2,t_2)\end{array}}{\Gamma \vdash \text{letrec } f_1\ x_1 = t_1 \sim \text{letrec } f_2\ x_2 = t_2 :: \Pi(x :: T).U}$
To show:
$|\Gamma| \mid \llbracket \Gamma \rrbracket \vdash \text{letrec } f_1\ x_1 = t_1 : |\Pi(x :: T).U| \sim \text{letrec } f_2\ x_2 = t_2 : |\Pi(x :: T).U| \mid \llbracket \Pi(x :: T).U \rrbracket(\mathbf{r}_1,\mathbf{r}_2)$
Expanding the definitions:
$|\Gamma| \mid \llbracket \Gamma \rrbracket \vdash \text{letrec } f_1\ x_1 = t_1 : |T| \to |U| \sim \text{letrec } f_2\ x_2 = t_2 : |T| \to |U| \mid \forall x_1 x_2. \llbracket T \rrbracket(x_1,x_2) \Rightarrow \llbracket U \rrbracket(\mathbf{r}_1 x_1,\ \mathbf{r}_2 x_2)$
By induction hypothesis on the premise:
$|\Gamma|, x_1, x_2 : |T|, f_1, f_2 : |T| \to |U| \mid \llbracket \Gamma \rrbracket, \llbracket T \rrbracket(x_1,x_2), \forall y_1, y_2. (\llbracket T \rrbracket(y_1,y_2) \wedge (y_1,y_2) < (x_1,x_2)) \Rightarrow \llbracket U \rrbracket(f_1 x_1,\ f_2 x_2) \vdash t_1 : |U| \sim t_2 : |U| \mid \llbracket U \rrbracket(\mathbf{r}_1,\mathbf{r}_2)$
And we apply the [LETREC] rule to get the result.

C.6 Proof of Lemma 16

By induction on the derivation of $\tau \searrow \ell$.

Case. $\dfrac{\ell \sqsubseteq \ell'}{\mathbb{T}_{\ell'}(\tau) \searrow \ell}$
Since $\ell \not\sqsubseteq a$ (given) and $\ell \sqsubseteq \ell'$ (premise), it must be the case that $\ell' \not\sqsubseteq a$. Hence, by definition, $\lfloor \mathbb{T}_{\ell'}(\tau) \rfloor_a(x,y) = \top$.

Case. $\dfrac{\tau \searrow \ell}{\mathbb{T}_{\ell'}(\tau) \searrow \ell}$
We consider two cases:

If $\ell' \not\sqsubseteq a$, then $\lfloor \mathbb{T}_{\ell'}(\tau) \rfloor_a(x,y) = \top$ by definition.

If $\ell' \sqsubseteq a$, then $\lfloor \mathbb{T}_{\ell'}(\tau) \rfloor_a(x,y) = \lfloor \tau \rfloor_a(x,y)$ by definition. By i.h. on the premise, we have $\lfloor \tau \rfloor_a(x,y) \equiv \top$. Composing, $\lfloor \mathbb{T}_{\ell'}(\tau) \rfloor_a(x,y) \equiv \top$.

Case. $\dfrac{\tau_1 \searrow \ell \qquad \tau_2 \searrow \ell}{\tau_1 \times \tau_2 \searrow \ell}$

By i.h. on the premises, we have $\lfloor \tau_i \rfloor_a(x,y) \equiv \top$ for $i = 1,2$ and all x,y. Therefore, $\lfloor \tau_1 \times \tau_2 \rfloor_a(x,y) \triangleq \lfloor \tau_1 \rfloor_a(\pi_1(x), \pi_1(y)) \wedge \lfloor \tau_2 \rfloor_a(\pi_2(x), \pi_2(y)) \equiv \top \wedge \top \equiv \top$.

Case. $\dfrac{\tau_2 \searrow \ell}{\tau_1 \to \tau_2 \searrow \ell}$

By i.h. on the premise, we have $\lfloor \tau_2 \rfloor_a(x,y) \equiv \top$ for all x,y. Hence, $\lfloor \tau_1 \to \tau_2 \rfloor_a(x,y) \triangleq (\forall v,w.\, \lfloor \tau_1 \rfloor_a(v,w) \Rightarrow \lfloor \tau_2 \rfloor_a(x\ v, y\ w)) \equiv (\forall v,w.\, \lfloor \tau_1 \rfloor_a(v,w) \Rightarrow \top) \equiv \top$.

C.7 Proof of Theorem 17

By induction on the given derivation of $\Gamma \vdash e : \tau$.

Case. $\dfrac{}{\Gamma \vdash \mathsf{tt} : \mathbb{B}}$

To show: $|\Gamma| \mid \lfloor \Gamma \rfloor_a \vdash \mathsf{tt} : \mathbb{B} \sim \mathsf{tt} : \mathbb{B} \mid (\mathbf{r}_1 = \mathsf{tt} \wedge \mathbf{r}_2 = \mathsf{tt}) \vee (\mathbf{r}_1 = \mathsf{ff} \wedge \mathbf{r}_2 = \mathsf{ff})$.
By rule TRUE, it suffices to show $(\mathsf{tt} = \mathsf{tt} \wedge \mathsf{tt} = \mathsf{tt}) \vee (\mathsf{tt} = \mathsf{ff} \wedge \mathsf{tt} = \mathsf{ff})$ in HOL, which is trivial.

Case. $\dfrac{\Gamma \vdash e : \mathbb{B} \qquad \Gamma \vdash e_t : \tau \qquad \Gamma \vdash e_f : \tau}{\Gamma \vdash \mathsf{case}\ e\ \mathsf{of}\ \mathsf{tt} \mapsto e_t; \mathsf{ff} \mapsto e_f : \tau}$

To show: $|\Gamma| \mid \lfloor \Gamma \rfloor_a \vdash (\mathsf{case}\ |e|_1\ \mathsf{of}\ \mathsf{tt} \mapsto |e_t|_1; \mathsf{ff} \mapsto |e_f|_1) : |\tau| \sim (\mathsf{case}\ |e|_2\ \mathsf{of}\ \mathsf{tt} \mapsto |e_t|_2; \mathsf{ff} \mapsto |e_f|_2) : |\tau| \mid \lfloor \tau \rfloor_a(\mathbf{r}_1, \mathbf{r}_2)$.
By i.h. on the first premise:
$|\Gamma| \mid \lfloor \Gamma \rfloor_a \vdash |e|_1 : \mathbb{B} \sim |e|_2 : \mathbb{B} \mid (\mathbf{r}_1 = \mathsf{tt} \wedge \mathbf{r}_2 = \mathsf{tt}) \vee (\mathbf{r}_1 = \mathsf{ff} \wedge \mathbf{r}_2 = \mathsf{ff})$
By i.h. on the second premise:
$|\Gamma| \mid \lfloor \Gamma \rfloor_a \vdash |e_t|_1 : |\tau| \sim |e_t|_2 : |\tau| \mid \lfloor \tau \rfloor_a(\mathbf{r}_1, \mathbf{r}_2)$
By i.h. on the third premise:
$|\Gamma| \mid \lfloor \Gamma \rfloor_a \vdash |e_f|_1 : |\tau| \sim |e_f|_2 : |\tau| \mid \lfloor \tau \rfloor_a(\mathbf{r}_1, \mathbf{r}_2)$
Applying rule BOOLCASE to the past three statements yields the required result.

Case. $\dfrac{}{\Gamma, x : \tau \vdash x : \tau}$

To show: $|\Gamma|, x_1 : |\tau|, x_2 : |\tau| \mid \lfloor \Gamma \rfloor_a, \lfloor \tau \rfloor_a(x_1, x_2) \vdash x_1 : |\tau| \sim x_2 : |\tau| \mid \lfloor \tau \rfloor_a(\mathbf{r}_1, \mathbf{r}_2)$.
This follows immediately from rule VAR.

Case. $\dfrac{\Gamma, x : \tau_1 \vdash e : \tau_2}{\Gamma \vdash \lambda x.e : \tau_1 \to \tau_2}$

To show: $|\Gamma| \mid \lfloor \Gamma \rfloor_a \vdash \lambda x_1.|e|_1 : |\tau_1| \to |\tau_2| \sim \lambda x_2.|e|_2 : |\tau_1| \to |\tau_2| \mid \forall x_1, x_2.\, \lfloor \tau_1 \rfloor_a(x_1, x_2) \Rightarrow \lfloor \tau_2 \rfloor_a(\mathbf{r}_1\ x_1, \mathbf{r}_2\ x_2)$.

By i.h. on the premise: $|\Gamma|, x_1 : |\tau_1|, x_2 : |\tau_2| \mid \lfloor\Gamma\rfloor_a, \lfloor\tau_1\rfloor_a(x_1,x_2) \vdash |e|_1 : |\tau_2| \sim |e|_2 : |\tau_2| \mid \lfloor\tau_2\rfloor_a(\mathbf{r}_1,\mathbf{r}_2)$.
Applying rule ABS immediately yields the required result.

Case. $\dfrac{\Gamma \vdash e : \tau_1 \to \tau_2 \qquad \Gamma \vdash e' : \tau_1}{\Gamma \vdash e\ e' : \tau_2}$
To show: $|\Gamma| \mid \lfloor\Gamma\rfloor_a \vdash |e|_1\ |e'|_1 : |\tau_2| \sim |e|_2\ |e'|_2 : |\tau_2| \mid \lfloor\tau_2\rfloor_a(\mathbf{r}_1,\mathbf{r}_2)$.
By i.h. on the first premise:
$|\Gamma| \mid \lfloor\Gamma\rfloor_a \vdash |e|_1 : |\tau_1| \to |\tau_2| \sim |e|_2 : |\tau_1| \to |\tau_2| \mid \forall x_1, x_2.\, \lfloor\tau_1\rfloor_a(x_1,x_2) \Rightarrow \lfloor\tau_2\rfloor_a(\mathbf{r}_1\ x_1, \mathbf{r}_2\ x_2)$
By i.h. on the second premise:
$|\Gamma| \mid \lfloor\Gamma\rfloor_a \vdash |e'|_1 : |\tau_1| \sim |e'|_2 : |\tau_1| \mid \lfloor\tau_1\rfloor_a(\mathbf{r}_1,\mathbf{r}_2)$
Applying rule APP immediately yields the required result.

Case. $\dfrac{\Gamma \vdash e : \tau \qquad \Gamma \vdash e' : \tau'}{\Gamma \vdash \langle e, e'\rangle : \tau \times \tau'}$
To show: $|\Gamma| \mid \lfloor\Gamma\rfloor_a \vdash \langle |e|_1, |e'|_1\rangle : |\tau| \times |\tau'| \sim \langle |e|_2, |e'|_2\rangle : |\tau| \times |\tau'| \mid \lfloor\tau\rfloor_a(\pi_1(\mathbf{r}_1), \pi_1(\mathbf{r}_2)) \wedge \lfloor\tau'\rfloor_a(\pi_2(\mathbf{r}_1), \pi_2(\mathbf{r}_2))$.
By i.h. on the first premise:
$|\Gamma| \mid \lfloor\Gamma\rfloor_a \vdash |e|_1 : |\tau| \sim |e|_2 : |\tau| \mid \lfloor\tau\rfloor_a(\mathbf{r}_1,\mathbf{r}_2)$
By i.h. on the second premise:
$|\Gamma| \mid \lfloor\Gamma\rfloor_a \vdash |e'|_1 : |\tau'| \sim |e'|_2 : |\tau'| \mid \lfloor\tau'\rfloor_a(\mathbf{r}_1,\mathbf{r}_2)$
The required result follows from the rule PAIR. We only need to show the third premise of the rule, i.e., the following in HOL:

$$\forall x_1 x_2 y_1 y_2.\, \lfloor\tau\rfloor_a(x_1,x_2) \Rightarrow \lfloor\tau'\rfloor_a(y_1,y_2) \Rightarrow (\lfloor\tau\rfloor_a(\pi_1\langle x_1,y_1\rangle, \pi_1\langle x_2,y_2\rangle) \wedge \lfloor\tau'\rfloor_a(\pi_2\langle x_1,y_1\rangle, \pi_2\langle x_2,y_2\rangle))$$

Since $\pi_1\langle x_1, y_1\rangle = x_1$, etc., this implication simplifies to:

$$\forall x_1 x_2 y_1 y_2.\, \lfloor\tau\rfloor_a(x_1,x_2) \Rightarrow \lfloor\tau'\rfloor_a(y_1,y_2) \Rightarrow (\lfloor\tau\rfloor_a(x_1,x_2) \wedge \lfloor\tau'\rfloor_a(y_1,y_2))$$

which is an obvious tautology.

Case. $\dfrac{\Gamma \vdash e : \tau \times \tau'}{\Gamma \vdash \pi_1(e) : \tau}$
To show: $|\Gamma| \mid \lfloor\Gamma\rfloor_a \vdash \pi_1(|e|_1) : |\tau| \sim \pi_1(|e|_2) : |\tau| \mid \lfloor\tau\rfloor_a(\mathbf{r}_1,\mathbf{r}_2)$.
By i.h. on the premise:
$|\Gamma| \mid \lfloor\Gamma\rfloor_a \vdash |e|_1 : |\tau| \times |\tau'| \sim |e|_2 : |\tau| \times |\tau'| \mid \lfloor\tau\rfloor_a(\pi_1(\mathbf{r}_1), \pi_1(\mathbf{r}_2)) \wedge \lfloor\tau'\rfloor_a(\pi_2(\mathbf{r}_1), \pi_2(\mathbf{r}_2))$
By rule SUB:
$|\Gamma| \mid \lfloor\Gamma\rfloor_a \vdash |e|_1 : |\tau| \times |\tau'| \sim |e|_2 : |\tau| \times |\tau'| \mid \lfloor\tau\rfloor_a(\pi_1(\mathbf{r}_1), \pi_1(\mathbf{r}_2))$
By rule PROJ$_1$, we get the required result.

Case. $\dfrac{\Gamma \vdash e : \tau}{\Gamma \vdash \eta_\ell(e) : \mathbb{T}_\ell(\tau)}$
To show: $|\Gamma| \mid \lfloor\Gamma\rfloor_a \vdash |e|_1 : |\tau| \sim |e|_2 : |\tau| \mid \lfloor\mathbb{T}_\ell(\tau)\rfloor_a(\mathbf{r}_1,\mathbf{r}_2)$.
By i.h. on the premise: $|\Gamma| \mid \lfloor\Gamma\rfloor_a \vdash |e|_1 : |\tau| \sim |e|_2 : |\tau| \mid \lfloor\tau\rfloor_a(\mathbf{r}_1,\mathbf{r}_2)$ (1)
If $\ell \sqsubseteq a$, then $\lfloor\mathbb{T}_\ell(\tau)\rfloor_a(\mathbf{r}_1,\mathbf{r}_2) \triangleq \lfloor\tau\rfloor_a(\mathbf{r}_1,\mathbf{r}_2)$, so the required result is the same as (1).

If $\ell \not\sqsubseteq a$, then $\lfloor \mathbb{T}_\ell(\tau) \rfloor_a(\mathbf{r}_1, \mathbf{r}_2) \triangleq \top$ and the required result follows from rule SUB applied to (1).

Case. $\dfrac{\Gamma \vdash e : \mathbb{T}_\ell(\tau) \qquad \Gamma, x : \tau \vdash e' : \tau' \qquad \tau' \searrow \ell}{\Gamma \vdash \mathsf{bind}(e, x.e') : \tau'}$

To show: $|\Gamma| \mid \lfloor \Gamma \rfloor_a \vdash (\lambda x.|e'|_1)\ |e|_1 : |\tau'| \sim (\lambda x.|e'|_2)\ |e|_2 : |\tau'| \mid \lfloor \tau' \rfloor_a(\mathbf{r}_1, \mathbf{r}_2)$.
By i.h. on the first premise:
$|\Gamma| \mid \lfloor \Gamma \rfloor_a \vdash |e|_1 : |\tau| \sim |e|_2 : |\tau| \mid \lfloor \mathbb{T}_\ell(\tau) \rfloor_a(\mathbf{r}_1, \mathbf{r}_2)$ (1)
By i.h. on the second premise:
$|\Gamma|, x_1 : |\tau|, x_2 : |\tau| \mid \lfloor \Gamma \rfloor_a, \lfloor \tau \rfloor_a(x_1, x_2) \vdash |e'|_1 : |\tau'| \sim |e'|_2 : |\tau'| \mid \lfloor \tau' \rfloor_a(\mathbf{r}_1, \mathbf{r}_2)$ (2)
We consider two cases:
Subcase. $\ell \sqsubseteq a$. Here, $\lfloor \mathbb{T}_\ell(\tau) \rfloor_a(\mathbf{r}_1, \mathbf{r}_2) \triangleq \lfloor \tau \rfloor_a(\mathbf{r}_1, \mathbf{r}_2)$, so (1) can be rewritten to:
$|\Gamma| \mid \lfloor \Gamma \rfloor_a \vdash |e|_1 : |\tau| \sim |e|_2 : |\tau| \mid \lfloor \tau \rfloor_a(\mathbf{r}_1, \mathbf{r}_2)$ (3)
Applying rule ABS to (2) yields:
$|\Gamma| \mid \lfloor \Gamma \rfloor_a \vdash \lambda x_1.|e'|_1 : |\tau| \to |\tau'| \sim \lambda x_2.|e'|_2 : |\tau| \to |\tau'| \mid \forall x_1 x_2. \lfloor \tau \rfloor_a(x_1, x_2) \Rightarrow \lfloor \tau' \rfloor_a(\mathbf{r}_1\ x_1, \mathbf{r}_2\ x_2)$
(4)
Applying rule APP to (4) and (3) yields:
$|\Gamma| \mid \lfloor \Gamma \rfloor_a \vdash (\lambda x_1.|e'|_1)\ |e|_1 : |\tau'| \sim (\lambda x_2.|e'|_2)\ |e|_2 : |\tau'| \mid \lfloor \tau' \rfloor_a(\mathbf{r}_1, \mathbf{r}_2)$
which is what we wanted to prove.
Subcase. $\ell \not\sqsubseteq a$. Here, $\lfloor \mathbb{T}_\ell(\tau) \rfloor_a(\mathbf{r}_1, \mathbf{r}_2) \triangleq \lfloor \tau \rfloor_a(\mathbf{r}_1, \mathbf{r}_2)$, so (1) can be rewritten to:
$|\Gamma| \mid \lfloor \Gamma \rfloor_a \vdash |e|_1 : |\tau| \sim |e|_2 : |\tau| \mid \top$ (5)
Applying rule ABS to (2) yields:
$|\Gamma| \mid \lfloor \Gamma \rfloor_a \vdash \lambda x_1.|e'|_1 : |\tau| \to |\tau'| \sim \lambda x_2.|e'|_2 : |\tau| \to |\tau'| \mid \forall x_1 x_2. \lfloor \tau \rfloor_a(x_1, x_2) \Rightarrow \lfloor \tau' \rfloor_a(\mathbf{r}_1\ x_1, \mathbf{r}_2\ x_2)$
By Lemma 16 applied to the subcase assumption $\ell \not\sqsubseteq a$ and the premise $\tau' \searrow \ell$, we have $\lfloor \tau' \rfloor_a(\mathbf{r}_1\ x_1, \mathbf{r}_2\ x_2) \equiv \top$. So, by rule SUB:
$|\Gamma| \mid \lfloor \Gamma \rfloor_a \vdash \lambda x_1.|e'|_1 : |\tau| \to |\tau'| \sim \lambda x_2.|e'|_2 : |\tau| \to |\tau'| \mid \forall x_1 x_2. \lfloor \tau \rfloor_a(x_1, x_2) \Rightarrow \top$
Since $(\forall x_1 x_2. \lfloor \tau \rfloor_a(x_1, x_2) \Rightarrow \top) \equiv \top \equiv (\forall x_1, x_2. \top \Rightarrow \top)$, we can use SUB again to get:
$|\Gamma| \mid \lfloor \Gamma \rfloor_a \vdash \lambda x_1.|e'|_1 : |\tau| \to |\tau'| \sim \lambda x_2.|e'|_2 : |\tau| \to |\tau'| \mid \forall x_1, x_2. \top \Rightarrow \top$ (6)
Applying rule APP to (6) and (5) yields:
$|\Gamma| \mid \lfloor \Gamma \rfloor_a \vdash (\lambda x_1.|e'|_1)\ |e|_1 : |\tau'| \sim (\lambda x_2.|e'|_2)\ |e|_2 : |\tau'| \mid \top$
which is the same as our goal since $\lfloor \tau' \rfloor_a(\mathbf{r}_1, \mathbf{r}_2) \equiv \top$.

C.8 Proof of Theorem 18

By induction on the derivation of $\Delta; \Phi; \Omega \vdash^l_k t : A$. We will show few cases.

Case. $\dfrac{}{\Delta; \Phi_a; \Omega, x : A \vdash^0_0 x : A}$

We can conclude by the following derivation:

$$\dfrac{\dfrac{}{(\!|\Omega|\!), x : (\!|A|\!)_v, \Delta \mid \Phi_a, \lfloor\Omega\rfloor, \lfloor A\rfloor_v(x) \vdash x : (\!|A|\!)_v \mid \lfloor A\rfloor_v(\mathbf{r})}\,\text{VAR} \qquad \dfrac{}{(\!|\Omega|\!), x : (\!|A|\!)_v, \Delta \mid \Phi_a, \lfloor\Omega\rfloor, \lfloor A\rfloor_v(x) \vdash 0 : \mathbb{N} \mid 0 \le \mathbf{r} \le 0}\,\textsc{Nat}}{(\!|\Omega|\!), x : (\!|A|\!)_v, \Delta \mid \Phi_a, \lfloor\Omega\rfloor, \lfloor A\rfloor_v(x) \vdash (x, 0) : (\!|A|\!)_v \times \mathbb{N} \mid \lfloor A\rfloor_v(\pi_1\mathbf{r}) \wedge 0 \le \pi_2\mathbf{r} \le 0}\,\text{PAIR-L}$$

where the additional proof conditions that is needed for the [PAIR-L] rule can be easily proved in HOL.

Case. $\dfrac{}{\Delta; \Phi_a; \Omega \vdash_0^0 \mathtt{n} : \mathrm{int}}$
Then we can conclude by the following derivation:

$$\dfrac{\dfrac{}{(\!|\Omega|\!), \Delta \mid \Phi_a, \lfloor\Omega\rfloor \vdash \mathtt{n} : \mathbb{N} \mid \top}\,\textsc{Nat} \qquad \dfrac{}{(\!|\Omega|\!), \Delta \mid \Phi_a, \lfloor\Omega\rfloor \vdash 0 : \mathbb{N} \mid 0 \le \mathbf{r} \le 0}\,\textsc{Nat}}{(\!|\Omega|\!), \Delta \mid \Phi_a, \lfloor\Omega\rfloor \vdash (\mathtt{n}, 0) : \mathbb{N} \times \mathbb{N} \mid 0 \le \pi_2\mathbf{r} \le 0}\,\text{PAIR-L}$$

where the additional proof conditions that is needed for the [PAIR-L] rule can be easily proved in HOL.

Case. $\dfrac{\Delta; \Phi_a; x : A_1, \Omega \vdash_k^l t : A_2}{\Delta; \Phi_a; \Omega \vdash_0^0 \lambda x.t : A_1 \xrightarrow{\mathrm{exec}(k,l)} A_2}$
By induction hypothesis we have $(\!|\Omega|\!), x : (\!|A_1|\!)_v, \Delta \mid \Phi, \lfloor\Omega\rfloor, \lfloor A_1\rfloor_v(x) \vdash (\!|t|\!) : (\!|A_2|\!)_e \mid \lfloor A\rfloor_e^{k,l}(\mathbf{r})$ and we can conclude by the following derivation:

$$\dfrac{\dfrac{\begin{array}{c}(\!|\Omega|\!), x : (\!|A_1|\!)_v, \Delta \mid \Phi, \lfloor\Omega\rfloor, \lfloor A_1\rfloor_v(x) \vdash (\!|t|\!) : \\ (\!|A_2|\!)_e \mid \lfloor A_2\rfloor_e^{k,l}(\mathbf{r})\end{array}}{\begin{array}{c}(\!|\Omega|\!), \Delta \mid \Phi, \lfloor\Omega\rfloor \vdash \lambda x.(\!|t|\!) : (\!|A_1|\!)_v \to (\!|A_2|\!)_e \mid \\ \forall x. \lfloor A_1\rfloor_v(x) \Rightarrow \lfloor A_2\rfloor_e^{k,l}(\mathbf{r}x)\end{array}}\,\text{ABS} \qquad (\!|\Omega|\!), \Delta \mid \Phi, \lfloor\Omega\rfloor \vdash 0 : \mathbb{N} \mid 0 \le \mathbf{r} \le 0}{(\!|\Omega|\!), \Delta \mid \Phi, \lfloor\Omega\rfloor \vdash (\lambda x.(\!|t|\!), 0) : ((\!|A_1|\!)_v \to (\!|A_2|\!)_e) \times \mathbb{N} \mid \forall x. \lfloor A_1\rfloor_v(x) \Rightarrow \lfloor A_2\rfloor_e^{k,l}((\pi_1\mathbf{r})x) \wedge 0 \le \pi_2\mathbf{r} \le 0}\,\text{PAIR-L}$$

where the additional proof conditions that is needed for the [PAIR-L] rule can be easily proved in HOL.

Case $\dfrac{\Delta; \Phi_a; \Omega \vdash_{k_1}^{l_1} t_1 : A_1 \xrightarrow{\mathrm{exec}(k,l)} A_2 \qquad \Delta; \Phi_a; \Omega \vdash_{k_2}^{l_2} t_2 : A_1}{\Delta; \Phi_a; \Omega \vdash_{k_1+k_2+k+c_{app}}^{l_1+l_2+l+c_{app}} t_1\, t_2 : A_2}$
By induction hypothesis and unfolding some some definitions we have

$$\begin{aligned}&(\!|\Omega|\!), \Delta \mid \Phi_a, \lfloor\Omega\rfloor \vdash (\!|t_1|\!) : ((\!|A_1|\!)_v \to ((\!|A_2|\!)_v \times \mathbb{N})) \times \mathbb{N} \mid \\ &\forall h. \lfloor A_1\rfloor_v(h) \Rightarrow (\lfloor A_2\rfloor_v(\pi_1((\pi_1(\mathbf{r}))h)) \wedge k \le \pi_2((\pi_1(\mathbf{r}))h) \le l) \wedge k_1 \le \pi_2(\mathbf{r}) \le l_1\end{aligned}$$

and $(\!|\Omega|\!), \Delta \mid \Phi_a, \lfloor\Omega\rfloor \vdash (\!|t_2|\!) : (\!|A_1|\!)_v \times \mathbb{N} \mid \lfloor A_1\rfloor_v(\pi_1(\mathbf{r})) \wedge k_2 \le \pi_2(\mathbf{r}) \le l_2$. So, we can prove:

$$\begin{aligned}&(\!|\Omega|\!), \Delta \mid \Phi_a, \lfloor\Omega\rfloor \vdash \mathsf{let}\, x = (\!|t_1|\!)\, \mathsf{in}\, \mathsf{let}\, y = (\!|t_2|\!)\, \mathsf{in}\, \pi_1(x)\, \pi_1(y) : (\!|A_2|\!)_v \times \mathbb{N} \mid \\ &\qquad\qquad \lfloor A_2\rfloor_v(\pi_1(\mathbf{r})) \wedge k \le \pi_2(\mathbf{r}) \le l \wedge k_1 \le \pi_2(x) \le l_1 \wedge k_2 \le \pi_2(y)\mathbf{r} \le l_2\end{aligned}$$

This combined with the definition of the cost-passing translation $(\!|t_1\, t_2|\!) \triangleq \mathsf{let}\, x = (\!|t_1|\!)\, \mathsf{in}\, \mathsf{let}\, y = (\!|t_2|\!)\, \mathsf{in}\, \mathsf{let}\, z = \pi_1(x)\ \ \pi_1(y)\, \mathsf{in}\, (\pi_1(z), \pi_2(x) + \pi_2(y) + \pi_2(z) + c_{app})$ allows us to prove as required the following:

$$(\!|\|\Omega\||\!), \Delta \mid \Phi_a, \lfloor \Omega \rfloor \vdash (\!|t_1\, t_2|\!) : (\!|\|A_2\||\!)_v \times \mathbb{N} \mid \\ \lfloor A_2 \rfloor_v(\pi_1(\mathbf{r})) \wedge k + k_1 + k_2 + c_{app} \leq \pi_2(\mathbf{r}) \leq l + l_1 + l_2 + c_{app}.$$

C.9 Proof of Theorem 19

To prove Theorem 19, we need three lemmas.

Lemma 33
Suppose $\Delta; \Phi \vdash \tau\, \mathsf{wf}$.[4] Then, the following hold:

1. $\Delta \mid \Phi \vdash \forall xy.\, \llbracket \tau \rrbracket_v(x,y) \Rightarrow \lfloor \overline{\tau} \rfloor_v(x) \wedge \lfloor \overline{\tau} \rfloor_v(y)$
2. $\Delta \mid \Phi \vdash \forall xy.\, \llbracket \tau \rrbracket_e^t(x,y) \Rightarrow \lfloor \overline{\tau} \rfloor_e^{0,\infty}(x) \wedge \lfloor \overline{\tau} \rfloor_e^{0,\infty}(y)$

Also, (3) $\llbracket \Gamma \rrbracket \Rightarrow \lfloor \overline{\Gamma}_1 \rfloor \wedge \lfloor \overline{\Gamma}_2 \rfloor$ where $\overline{\Gamma}_1$ and $\overline{\Gamma}_2$ are obtained by replacing each variable x in $\overline{\Gamma}$ with x_1 and x_2, respectively.

Proof
(1) and (2) follow by a simultaneous induction on the given judgment. (3) follows immediately from (1). □

Lemma 34
If $\Delta; \Phi_a; \Gamma \vdash e_1 \ominus e_2 \lesssim t : \tau$ in RelCost, then $\Delta; \Phi; \overline{\Gamma} \vdash_0^\infty e_i : \overline{\tau}$ for $i \in \{1,2\}$ in RelCost.

Proof
By induction on the given derivation. □

Lemma 35
If $\Delta; \Phi \models \tau_1 \sqsubseteq \tau_2$, then $\Delta; \Phi \vdash \forall xy.\, \llbracket \tau_1 \rrbracket_v(x,y) \Rightarrow \llbracket \tau_2 \rrbracket_v(x,y)$.

Proof
By induction on the given derivation of $\Delta; \Phi \models \tau_1 \sqsubseteq \tau_2$. □

Proof of Theorem 19
The proof is by induction on the given derivation of $\Delta; \Phi; \Gamma \vdash t_1 \ominus t_2 \lesssim k : \tau$. We show only a few representative cases here.

Case: $$\dfrac{i :: S, \Delta; \Phi_a; \Gamma \vdash e \ominus e' \lesssim t : \tau \qquad i \notin \mathrm{FIV}(\Phi_a; \Gamma)}{\Delta; \Phi_a; \Gamma \vdash \Lambda e \ominus \Lambda e' \lesssim 0 : \forall i \overset{\mathrm{diff}(t)}{::} S.\, \tau}\ \textsc{r-iLam}$$

To show: $\|\Gamma\|, \Delta \mid \Phi_a, \llbracket \Gamma \rrbracket \vdash (\lambda\_.(\!|e|\!)_1, 0) : (\mathbb{N} \to (\!|\|\tau\||\!)_e) \times \mathbb{N} \sim (\lambda\_.(\!|e'|\!)_2, 0) : (\mathbb{N} \to (\!|\|\tau\||\!)_e) \times \mathbb{N} \mid \llbracket \forall i \overset{\mathrm{diff}(t)}{::} S.\, \tau \rrbracket_e^0(\mathbf{r}_1, \mathbf{r}_2)$.

Expand $\llbracket \forall i \overset{\mathrm{diff}(t)}{::} S.\, \tau \rrbracket_e^0(\mathbf{r}_1, \mathbf{r}_2)$ to $\llbracket \forall i \overset{\mathrm{diff}(t)}{::} S.\, \tau \rrbracket_v(\pi_1\ \mathbf{r}_1, \pi_1\ \mathbf{r}_2) \wedge \pi_2 \mathbf{r}_1 - \pi_2\ \mathbf{r}_2 \leq 0$, and apply the rule [PAIR] to reduce to two proof obligations:

[4] This judgment simply means that τ is well-formed in the context $\Delta; \Phi$. It is defined in the original RelCost paper (Çiçek *et al.*, 2017).

(A) $\|\Gamma\|, \Delta \mid \Phi_a, \llbracket\Gamma\rrbracket \vdash \lambda\_.(\!|e|\!)_1 : \mathbb{N} \to (\!|\|\tau|\!|)_e \sim \lambda\_.(\!|e'|\!)_2 : \mathbb{N} \to (\!|\|\tau|\!|)_e \mid \llbracket\forall i \overset{\text{diff}(t)}{::} S.\,\tau\rrbracket_v(\mathbf{r}_1, \mathbf{r}_2)$
(B) $\|\Gamma\|, \Delta \mid \Phi_a, \llbracket\Gamma\rrbracket \vdash 0 : \mathbb{N} \sim 0 : \mathbb{N} \mid \mathbf{r}_1 - \mathbf{r}_2 \leq 0$

(B) follows immediately by rule [ZERO]. To prove (A), expand $\llbracket\forall i \overset{\text{diff}(t)}{::} S.\,\tau\rrbracket_v(\mathbf{r}_1, \mathbf{r}_2)$ and apply rule [$\wedge_\mathsf{I}$]. We get three proof obligations.

(C) $\|\Gamma\|, \Delta \mid \Phi_a, \llbracket\Gamma\rrbracket \vdash \lambda\_.(\!|e|\!)_1 : \mathbb{N} \to (\!|\|\tau|\!|)_e \sim \lambda\_.(\!|e'|\!)_2 : \mathbb{N} \to (\!|\|\tau|\!|)_e \mid \lfloor\forall i \overset{\text{exec}(0,\infty)}{::} S.\,\overline{\tau}\rfloor_v(\mathbf{r}_1)$

(D) $\|\Gamma\|, \Delta \mid \Phi_a, \llbracket\Gamma\rrbracket \vdash \lambda\_.(\!|e|\!)_1 : \mathbb{N} \to (\!|\|\tau|\!|)_e \sim \lambda\_.(\!|e'|\!)_2 : \mathbb{N} \to (\!|\|\tau|\!|)_e \mid \lfloor\forall i \overset{\text{exec}(0,\infty)}{::} S.\,\overline{\tau}\rfloor_v(\mathbf{r}_2)$

(E) $\|\Gamma\|, \Delta \mid \Phi_a, \llbracket\Gamma\rrbracket \vdash \lambda\_.(\!|e|\!)_1 : \mathbb{N} \to (\!|\|\tau|\!|)_e \sim \lambda\_.(\!|e'|\!)_2 : \mathbb{N} \to (\!|\|\tau|\!|)_e \mid \forall z_1 z_2. \top \Rightarrow \forall i. \llbracket\tau\rrbracket_e^t(\mathbf{r}_1\ z_1, \mathbf{r}_2\ z_2)$

To prove (C), apply Lemma 34 to the given derivation (not just the premise), obtaining a RelCost derivation for $\Delta; \Phi_a; \overline{\Gamma} \vdash_0^\infty \Lambda e : (\forall i \overset{\text{exec}(0,\infty)}{::} S.\,\overline{\tau})$. Applying Theorem 18 to this yields $(\!|\overline{\Gamma}|\!), \Delta \mid \Phi_a, \lfloor\overline{\Gamma}\rfloor \vdash (\lambda\_.(\!|e|\!), 0) : (\mathbb{N} \to (\!|\|\overline{\tau}|\!|)_e) \times \mathbb{N} \mid \lfloor\forall i \overset{\text{exec}(0,\infty)}{::} S.\,\overline{\tau}\rfloor_e^{0,\infty}(\mathbf{r})$ in UHOL, which is the same as $(\!|\overline{\Gamma}|\!), \Delta \mid \Phi_a, \lfloor\overline{\Gamma}\rfloor \vdash (\lambda\_.(\!|e|\!), 0) : (\mathbb{N} \to (\!|\|\overline{\tau}|\!|)_e) \times \mathbb{N} \mid \lfloor\forall i \overset{\text{exec}(0,\infty)}{::} S.\,\overline{\tau}\rfloor_v(\pi_1\ \mathbf{r}) \wedge 0 \leq \pi_2\ \mathbf{r} \leq \infty$. Applying rule [PROJ$_1$], we get $(\!|\overline{\Gamma}|\!), \Delta \mid \Phi_a, \lfloor\overline{\Gamma}\rfloor \vdash \pi_1(\lambda\_.(\!|e|\!), 0) : \mathbb{N} \to (\!|\|\overline{\tau}|\!|)_e \mid \lfloor\forall i \overset{\text{exec}(0,\infty)}{::} S.\,\overline{\tau}\rfloor_v(\mathbf{r})$. By subject conversion, $(\!|\overline{\Gamma}|\!), \Delta \mid \Phi_a, \lfloor\overline{\Gamma}\rfloor \vdash \lambda\_.(\!|e|\!) : \mathbb{N} \to (\!|\|\overline{\tau}|\!|)_e \mid \lfloor\forall i \overset{\text{exec}(0,\infty)}{::} S.\,\overline{\tau}\rfloor_v(\mathbf{r})$. Renaming variables, we get $(\!|\overline{\Gamma}|\!)_1, \Delta \mid \Phi_a, \lfloor\overline{\Gamma}_1\rfloor \vdash \lambda\_.(\!|e|\!)_1 : \mathbb{N} \to (\!|\|\overline{\tau}|\!|)_e \mid \lfloor\forall i \overset{\text{exec}(0,\infty)}{::} S.\,\overline{\tau}\rfloor_v(\mathbf{r})$.

Now note that by definition, $\|\Gamma\| \supseteq (\!|\overline{\Gamma}|\!)_1$ and by Lemma 33(3), $\llbracket\Gamma\rrbracket \Rightarrow \lfloor\overline{\Gamma}_1\rfloor$. Hence, we also get $\|\Gamma\|, \Delta \mid \Phi_a, \llbracket\Gamma\rrbracket \vdash \lambda\_.(\!|e|\!)_1 : \mathbb{N} \to (\!|\|\overline{\tau}|\!|)_e \mid \lfloor\forall i \overset{\text{exec}(0,\infty)}{::} S.\,\overline{\tau}\rfloor_v(\mathbf{r})$. (C) follows immediately by rule [UHOL-L].

(D) has a similar proof.

To prove (E), apply the rule [ABS], getting the obligation:
$\|\Gamma\|, \Delta, z_1, z_2 : \mathbb{N} \mid \Phi_a, \llbracket\Gamma\rrbracket \vdash (\!|e|\!)_1 : (\!|\|\tau|\!|)_e \sim (\!|e'|\!)_2 : (\!|\|\tau|\!|)_e \mid \forall i. \llbracket\tau\rrbracket_e^t(\mathbf{r}_1, \mathbf{r}_2)$
Since z_1, z_2 do not appear anywhere else, we can strengthen the context to remove them, thus reducing to: $\|\Gamma\|, \Delta \mid \Phi_a, \llbracket\Gamma\rrbracket \vdash (\!|e|\!)_1 : (\!|\|\tau|\!|)_e \sim (\!|e'|\!)_2 : (\!|\|\tau|\!|)_e \mid \forall i. \llbracket\tau\rrbracket_e^t(\mathbf{r}_1, \mathbf{r}_2)$
Next, we transpose to HOL using Theorem 6. We get the obligation:
$\|\Gamma\|, \Delta \mid \Phi_a, \llbracket\Gamma\rrbracket \vdash \forall i. \llbracket\tau\rrbracket_e^t((\!|e|\!)_1, (\!|e'|\!)_2)$
This is equivalent to:
$\|\Gamma\|, \Delta, i : S \mid \Phi_a, \llbracket\Gamma\rrbracket \vdash \llbracket\tau\rrbracket_e^t((\!|e|\!)_1, (\!|e'|\!)_2)$
The last statement follows immediately from i.h. on the premise, followed by transposition to HOL using Theorem 6.

Case: $$\frac{\Delta; \Phi_a; \Gamma \vdash e \ominus e \lesssim t : \tau \qquad \forall x \in dom(\Gamma).\ \ \Delta; \Phi_a \models \Gamma(x) \sqsubseteq \Box\Gamma(x)}{\Delta; \Phi_a; \Gamma, \Gamma'; \Omega \vdash e \ominus e \lesssim 0 : \Box\tau}\ \textsc{nochange}$$
To show: $\|\Gamma\|, \Delta \mid \Phi_a, \llbracket\Gamma\rrbracket \vdash (\!|e|\!)_1 : (\!|\|\tau|\!|)_e \sim (\!|e|\!)_2 : (\!|\|\tau|\!|)_e \mid \llbracket\Box\tau\rrbracket_e^0(\mathbf{r}_1, \mathbf{r}_2)$.
Expanding the definition of $\llbracket\Box\tau\rrbracket_e^0$, this is equivalent to:
$\|\Gamma\|, \Delta \mid \Phi_a, \llbracket\Gamma\rrbracket \vdash (\!|e|\!)_1 : (\!|\|\tau|\!|)_e \sim (\!|e|\!)_2 : (\!|\|\tau|\!|)_e \mid \llbracket\tau\rrbracket_v(\pi_1\ \mathbf{r}_1, \pi_2\ \mathbf{r}_2) \wedge (\pi_1\ \mathbf{r}_1 = \pi_1\ \mathbf{r}_2) \wedge (\pi_2\ \mathbf{r}_1 - \pi_2\ \mathbf{r}_2 \leq 0)$
Using rule [$\wedge_\mathsf{I}$], we reduce this to two obligations:
(A) $\|\Gamma\|, \Delta \mid \Phi_a, \llbracket\Gamma\rrbracket \vdash (\!|e|\!)_1 : (\!|\|\tau|\!|)_e \sim (\!|e|\!)_2 : (\!|\|\tau|\!|)_e \mid \llbracket\tau\rrbracket_v(\pi_1\ \mathbf{r}_1, \pi_2\ \mathbf{r}_2)$

(B) $\|\Gamma\|, \Delta \mid \Phi_a, \llbracket\Gamma\rrbracket \vdash (\!|e|\!)_1 : (\!|\|\tau\||\!)_e \sim (\!|e|\!)_2 : (\!|\|\tau\||\!)_e \mid (\pi_1\ \mathbf{r}_1 = \pi_1\ \mathbf{r}_2) \wedge (\pi_2\ \mathbf{r}_1 - \pi_2\ \mathbf{r}_2 \leq 0)$

By i.h. on the first premise,
$\|\Gamma\|, \Delta \mid \Phi_a, \llbracket\Gamma\rrbracket \vdash (\!|e|\!)_1 : (\!|\|\tau\||\!)_e \sim (\!|e|\!)_2 : (\!|\|\tau\||\!)_e \mid \llbracket\tau\rrbracket_v(\pi_1\ \mathbf{r}_1, \pi_2\ \mathbf{r}_2) \wedge (\pi_2\ \mathbf{r}_1 - \pi_2\ \mathbf{r}_2 \leq t)$
By rule [SUB],
$\|\Gamma\|, \Delta \mid \Phi_a, \llbracket\Gamma\rrbracket \vdash (\!|e|\!)_1 : (\!|\|\tau\||\!)_e \sim (\!|e|\!)_2 : (\!|\|\tau\||\!)_e \mid \llbracket\tau\rrbracket_v(\pi_1\ \mathbf{r}_1, \pi_2\ \mathbf{r}_2)$
which is the same as (A).

To prove (B), apply Lemma 35 to the second premise to get for every $x \in dom(\Gamma)$ that $\Delta \mid \Phi_a \vdash \llbracket\Gamma(x)\rrbracket_v(x_1, x_2) \Rightarrow \llbracket\Box\Gamma(x)\rrbracket_v(x_1, x_2)$. Since $\llbracket\Box\Gamma(x)\rrbracket_v(x_1, x_2) \Rightarrow x_1 = x_2$ and from $\llbracket\Gamma\rrbracket$ we know that $\llbracket\Gamma(x)\rrbracket_v(x_1, x_2)$, it follows that $\|\Gamma\|, \Delta \mid \Phi_a, \llbracket\Gamma\rrbracket \vdash x_1 = x_2$. Since this holds for every $x \in dom(\Gamma)$, it follows immediately that $\|\Gamma\|, \Delta \mid \Phi_a, \llbracket\Gamma\rrbracket \vdash (\!|e|\!)_1 = (\!|e|\!)_2$. By Theorem 6, $\|\Gamma\|, \Delta \mid \Phi_a, \llbracket\Gamma\rrbracket \vdash (\!|e|\!)_1 : (\!|\|\tau\||\!)_e \sim (\!|e|\!)_2 : (\!|\|\tau\||\!)_e \mid \mathbf{r}_1 = \mathbf{r}_2$. (B) follows immediately by rule [SUB].
$\Box$

D Examples

D.1 Factorial

This example shows that the two following implementations of factorial, with and without accumulator, are equivalent:

$$\text{fact}_1 \triangleq \text{letrec}\ f_1\ n_1 = \text{case}\ n_1\ \mathsf{of}\ 0 \mapsto 1; S \mapsto \lambda x_1.Sx_1 * (f_1\ x_1)$$
$$\text{fact}_2 \triangleq \text{letrec}\ f_2\ n_2 = \lambda acc.\text{case}\ n_2\ \mathsf{of}\ 0 \mapsto acc; S \mapsto \lambda x_2.f_2\ x_2\ (Sx_2 * acc)$$

Our goal is to prove that:

$$\emptyset \mid \emptyset \vdash \text{fact}_1 : \mathbb{N} \to \mathbb{N} \sim \text{fact}_2 : \mathbb{N} \to \mathbb{N} \to \mathbb{N} \mid \forall n_1 n_2. n_1 = n_2 \Rightarrow \forall acc. (\mathbf{r}_1\ n_1) * acc = \mathbf{r}_2\ n_2\ acc$$

Since both programs do the same number of iterations, we can do synchronous reasoning for the recursion at the head of the programs. However, the bodies of the functions have different types since fact_2 receives an extra argument, the accumulator. Therefore, we will need a one-sided application of [ABS-R], before we can go back to reasoning synchronously. We will then apply the [CASE] rule, knowing that both terms reduce to the same branch, since $n_1 = n_2$. On the zero branch, we will need to prove the trivial equality $1 * acc = acc$. On the successor branch, we will need to prove that $Sx * (\text{fact}\ x) * acc = \text{fact}_2\ x_2\ (Sx_2 * acc)$, knowing by induction hypothesis that such a property holds for every m less that x.

Now we will expand on the details. We start the proof applying the [LETREC] rule, which has 2 premises:

1. Both functions are well-defined
2. $n_1 = n_2, \forall y_1 y_2. (y_1, y_2) < (n_1, n_2) \Rightarrow y_1 = y_2 \Rightarrow \forall acc. (f_1\ y_1) * acc = f_2\ y_2\ acc \vdash$ $\text{case}\ n_1\ \mathsf{of}\ 0 \mapsto 1; S \mapsto \lambda x_1.Sx_1 * (f_1\ x_1) \sim \lambda acc.\text{case}\ n_2\ \mathsf{of}\ 0 \mapsto acc; S \mapsto$ $\lambda x_2.f_2\ x_2\ (Sx_2 * acc) \mid n_1 = n_2 \Rightarrow \forall acc.\mathbf{r}_1 * acc = \mathbf{r}_2\ acc$

We assume that the first premise is provable.

To prove the second premise, we start by applying ABS-R, which leaves the following proof obligation:

$$n_1 = n_2, \forall y_1 y_2.(y_1, y_2) < (n_1, n_2) \Rightarrow y_1 = y_2 \Rightarrow \forall acc.(f_1\ y_1) * acc = f_2\ y_2\ acc, n_1 = n_2 \vdash$$
$$\text{case } n_1 \text{ of } 0 \mapsto 1; S \mapsto \lambda x_1.Sx_1 * (f_1\ x_1) \sim \text{case } n_2 \text{ of } 0 \mapsto acc; S \mapsto \lambda x_2.f_2\ x_2\ (Sx_2 * acc) \mid \mathbf{r}_1 * acc = \mathbf{r}_2$$

Now we can apply [CASE], and we have 3 premises, where Ψ denotes the axioms of the previous judgment:

- $\Psi \vdash n_1 \sim n_2 \mid \mathbf{r}_1 = 0 \Leftrightarrow \mathbf{r}_2 = 0$
- $\Psi, n_1 = 0, n_2 = 0 \vdash 1 \sim acc \mid \mathbf{r}_1 * acc = \mathbf{r}_2$
- $\Psi \vdash \lambda x_1.Sx_1 * (f_1\ x_1) \sim \lambda x_2.f_2\ x_2\ (Sx_2 * acc) \mid \forall x_1 x_2.n_1 = Sx_1 \Rightarrow n_2 = Sx_2 \Rightarrow (\mathbf{r}_1\ x_1) * acc = \mathbf{r}_2\ x_2$

Premise 1 is a direct consequence of $n_1 = n_2$. Premise 2 is a trivial arithmetic identity. To prove premise 3, we first apply the ABS rule:

$$\Psi, n_1 = Sx_1, n_2 = Sx_2 \vdash Sx_1 * (f_1\ x_1) \sim f_2\ x_2\ (Sx_2 * acc) \mid \mathbf{r}_1 * acc = \mathbf{r}_2$$

and then by Theorem 6 we can finish the proof in HOL by deriving.

$$\Psi, n_1 = Sx_1, n_2 = Sx_2 \vdash Sx_1 * (f_1\ x_1) * acc = f_2\ x_2\ (Sx_2 * acc)$$

From the premises we can first prove that $(x_1, x_2) < (n_1, n_2)$ so by the inductive hypothesis from the [LETREC] rule, and the [$\Rightarrow_E$] rule, we get

$$\forall acc.(f_1\ x_1) * acc = f_2\ x_2\ acc,$$

which we then instantiate with $Sx_1 * acc$ to get

$$(f_1\ x_1) * Sx_1 * acc = f_2\ x_2\ (Sx_1 * acc).$$

On the other hand, from the hypotheses we also have $x_1 = x_2$, so by [CONV] we finally prove

$$(f_1\ x_1) * Sx_1 * acc = f_2\ x_2\ (Sx_2 * acc)$$

D.2 List reversal

A related example for lists is the equivalence of reversal with and without accumulator. The structure of the proof is the same as in the factorial example, but we will briefly show it to illustrate how the LISTCASE rule is used. The functions are written:

$$\begin{array}{ll} \text{rev}_1 & \triangleq \text{letrec } f_1\ l_1 = \text{case } l_1 \text{ of } [] \mapsto []; \_ :: \_ \mapsto \lambda h_1.\lambda t_1.(f_1\ t_1) ++ [x_1] \\ \text{rev}_2 & \triangleq \text{letrec } f_2\ l_2 = \lambda acc.\text{case } l_2 \text{ of } [] \mapsto acc; \_ :: \_ \mapsto \lambda h_2.\lambda t_2.f_2\ t_2\ (h_2 :: acc) \end{array}$$

We want to prove they are related by the following judgment:

$$\emptyset \mid \emptyset \vdash \text{rev}_1 : \mathsf{list}_\tau \to \mathsf{list}_\tau \sim \text{rev}_2 : \mathsf{list}_\tau \to \mathsf{list}_\tau \mid \forall l_1, l_2.l_1 = l_2 \Rightarrow \forall acc.\ (\mathbf{r}_1\ l_1) ++ acc = \mathbf{r}_2\ l_2\ acc$$

By the [LETREC] rule, we have to prove 2 premises:

1. Both functions are well-defined.
2. $l_1 = l_2, \forall m_1 m_2.(|m_1|, |m_2|) < (|l_1|, |l_2|) \Rightarrow m_1 = m_2 \Rightarrow \forall acc.(f_1\ m_1) ++ acc = f_2\ m_2\ acc \vdash \text{case } l_1 \text{ of } [] \mapsto []; \_ :: \_ \mapsto \lambda h_1.\lambda t_1.(f_1\ t_1) ++ [x_1] \sim \lambda acc.\text{case } l_2 \text{ of } [] \mapsto acc; \_ :: \_ \mapsto \lambda h_2.\lambda t_2.f_2\ t_2\ (h_2 :: acc) \mid \forall acc.\ \mathbf{r}_1 ++ acc = \mathbf{r}_2\ acc$

For the second premise, similarly as in factorial, we apply ABS-R. We have the following premise, where Ψ denotes the axioms in the previous judgment:

$$\begin{gathered}\Psi \vdash \text{case } l_1 \text{ of } [] \mapsto [];\_ :: \_ \mapsto \lambda h_1.\lambda t_1.(f_1\ t_1) +\!+ [x_1] \sim \\ \text{case } t_2 \text{ of } [] \mapsto acc;\_ :: \_ \mapsto \lambda h_2.\lambda t_2.f_2\ t_2\ (h_2 :: acc) \mid \\ \mathbf{r}_1 +\!+ acc = \mathbf{r}_2\end{gathered}$$

and then LISTCASE, which has three premises:

- $\Psi \vdash l_1 \sim l_2 \mid \mathbf{r}_1 = [] \Leftrightarrow \mathbf{r}_2 = []$
- $\Psi, l_1 = [], l_2 = [] \vdash [] \sim acc \mid \mathbf{r}_1 +\!+ acc = \mathbf{r}_2$
- $\Psi \vdash \lambda h_1.\lambda t_1.(f_1\ t_1) +\!+ [x_1] \sim \lambda h_2.\lambda t_2.f_2\ t_2\ (h_2 :: acc) \mid$
  $\forall h_1 t_1 h_2 t_2. l_1 = h_1 :: t_1 \Rightarrow l_2 = h_2 :: t_2 \Rightarrow \mathbf{r}_1 +\!+ acc = \mathbf{r}_2$

We complete the proof in a similar way as in the factorial example.

D.3 Proof of Theorem 23

We will use without proof two unary lemmas:

Lemma 36
$\bullet \mid \bullet \vdash take : \mathsf{list}_{\mathbb{N}} \to \mathbb{N} \to \mathsf{list}_{\mathbb{N}} \mid \forall ln.|r\ l\ n| = min(n, |l|)$

Lemma 37
$\bullet \mid \bullet \vdash map : \mathsf{list}_{\mathbb{N}} \to (\mathbb{N} \to \mathbb{N}) \to \mathsf{list}_{\mathbb{N}} \mid \forall lf.|r\ l\ f| = |l|$

Now we proceed with the proof of the theorem

We want to prove

$$\begin{gathered}l_1 = l_2, n_1 = n_2, g_1 = g_2 \vdash map\ (take\ l_1\ n_1)\ g_1 \sim take\ (map\ l_2\ g_2)\ n_2 \mid \\ \mathbf{r}_1 \sqsubseteq \mathbf{r}_2 \wedge |\mathbf{r}_1| = \mathsf{min}(n_1, |l_1|) \wedge |\mathbf{r}_2| = \mathsf{min}(n_2, |l_2|)\end{gathered}$$

where $\mathbf{r}_1 \sqsubseteq \mathbf{r}_2$ is the prefix ordering and is defined as an inductive predicate:

$$\forall l.[] \sqsubseteq l \qquad\qquad \forall h l_1 l_2. l_1 \sqsubseteq l_2 \Rightarrow h :: l_1 \sqsubseteq h :: l_2$$

By the helping lemmas and Lemma 10, it suffices to prove just the first conjunct:

$$l_1 = l_2, n_1 = n_2, g_1 = g_2 \vdash map\ (take\ l_1\ n_1)\ g_1 \sim take\ (map\ l_2\ g_2)\ n_2 \mid \mathbf{r}_1 \sqsubseteq \mathbf{r}_2$$

The derivation begins by applying the APP-R rule. We get the following judgment on n_2:

$$l_1 = l_2, n_1 = n_2, g_1 = g_2 \vdash n_2 \mid \mathbf{r} \geq |take\ l_1\ n_1| \tag{1}$$

and a main premise:

$$\begin{gathered}l_1 = l_2, n_1 = n_2, g_1 = g_2 \vdash map\ (take\ l_1\ n_1)\ g_1 \sim take\ (map\ l_2\ g_2) \mid \\ \forall x_2. x_2 \geq |take\ l_1\ n_1| \Rightarrow \mathbf{r}_1 \sqsubseteq (\mathbf{r}_2\ x_2)\end{gathered} \tag{2}$$

Notice that we have chosen the premise $x_2 \geq |take\ l_1\ n_1|$ because we are trying to prove $\mathbf{r}_1 \sqsubseteq (\mathbf{r}_2\ x_2)$, which is only true if we take a larger prefix on the right than on the left. The judgment (1) is easily proven from the fact that $|take\ l_1\ n_1| = min(n_1, |l_1|) \leq n_1 = n_2$, which we get from the lemmas. To prove (2) we first apply APP-L with a trivial condition $g_1 = g_2$ on g_1. Then we apply APP and we have two premises:

(A) $\Psi \vdash take\ l_1\ n_1 \sim map\ l_2\ g_2 \mid \mathbf{r}_1 \sqsubseteq_{g_2} \mathbf{r}_2$
(B) $\Psi \vdash map \sim take \mid$
$\forall m_1 m_2. m_1 \sqsubseteq_{g_2} m_2 \Rightarrow (\forall g_1. g_1 = g_2 \Rightarrow \forall x_2. x_2 \geq |m_1| \Rightarrow (\mathbf{r}_1\ m_1\ g_1) \sqsubseteq (\mathbf{r}_2\ m_2\ x_2))$

where $\sqsubseteq_g$ is defined as an inductive predicate parametrized by g:

$$\forall l. [] \sqsubseteq_g l \qquad \forall h l_1 l_2. l_1 \sqsubseteq_g l_2 \Rightarrow h :: l_1 \sqsubseteq_g (gh) :: l_2$$

We first show how to prove (A). We start by applying APP with a trivial condition for the arguments to get:

$$\Psi \vdash take\ l_1 \sim map\ l_2 \mid \forall x_1 g_2. (\mathbf{r}_1\ x_1) \sqsubseteq_{g_2} (\mathbf{r}_2\ g_2)$$

We then apply APP, which has two premises, one of them equating l_1 and l_2. The other one is:

$$\Psi \vdash take \sim map \mid \forall m_1 m_2. m_1 = m_2 \Rightarrow \forall x_1 g_2. (\mathbf{r}_1\ m_1\ x_1) \sqsubseteq_{g_2} (\mathbf{r}_2\ m_2\ g_2)$$

To complete this branch of the proof, we apply LETREC. We need to prove the following premise:

$$\Psi, m_1 = m_2, \forall k_1 k_2. (k_1, k_2) < (m_1, m_2) \Rightarrow k_1 = k_2 \Rightarrow \forall x_1 g_2. (f_1\ k_1\ x_1) \sqsubseteq_{g_2} (f_2\ k_2\ g_2) \vdash$$
$$\lambda n_1. e_1 \sim \lambda g_2. e_2 \mid \forall x_1 g_2. (\mathbf{r}_1\ x_1) \sqsubseteq_{g_2} (\mathbf{r}_2\ g_2)$$

Where e_1, e_2 abbreviate the bodies of the functions:

$$\begin{array}{rl} e_1 \triangleq & \text{case } m_1 \text{ of } [] \mapsto [] \\ & ;\_ :: \_ \mapsto \quad \lambda h_1 t_1. \text{case } x_1 \textsf{ of } 0 \mapsto \quad [] \\ & \qquad\qquad\qquad\qquad ;S \mapsto \quad \lambda y_1. h_1 :: f_1\ t_1\ y_1 \end{array}$$

$$\begin{array}{rl} e_2 \triangleq & \text{case } m_2 \text{ of } [] \mapsto \quad [] \\ & ;\_ :: \_ \mapsto \quad \lambda h_2 t_2. (g_2\ h_2) :: (f_2\ t_2\ g_2) \end{array}$$

If we apply ABS we get a premise:

$$\Psi, m_1 = m_2, \forall k_1 k_2. (k_1, k_2) < (m_1, m_2) \Rightarrow k_1 = k_2 \Rightarrow$$
$$\forall x_1 g_2. (f_1\ k_1\ x_1) \sqsubseteq_{g_2} (f_2\ k_2\ g_2) \vdash e_1 \sim e_2 \mid \mathbf{r}_1 \sqsubseteq_f \mathbf{r}_2$$

And now we can apply a synchronous CASE rule, since we have a premise $m_1 = m_2$. This yields 3 proof obligations, where Ψ' is the set of axioms in the previous judgment:

(A.1) $\Psi' \vdash m_1 \sim m_2 \mid \mathbf{r}_1 = [] \Leftrightarrow \mathbf{r}_2 = []$
(A.2) $\Psi' \vdash [] \sim [] \mid \mathbf{r}_1 \sqsubseteq_f \mathbf{r}_2$
(A.3) $\Psi' \vdash \lambda h_1 t_1. \text{case } x_1 \textsf{ of } 0 \mapsto []; S \mapsto \lambda y_1. h_1 :: f_1\ t_1\ y_1 \sim$
$\lambda h_2 t_2. (g_2\ h_2) :: (f_2\ t_2\ g_2) \mid \forall h_1 t_1 h_2 t_2. m_1 = h_1 :: t_1 \Rightarrow m_2 = h_2 :: t_2 \Rightarrow (\mathbf{r}_1\ h_1\ t_1) \sqsubseteq_{g_2}$
$(\mathbf{r}_2\ h_2\ t_2)$

Premises (A.1) and (A.2) are trivial. To prove (A.3) we first apply ABS twice:

$$\Psi', m_1 = h_1 :: t_1, m_2 = h_2 :: t_2 \vdash$$
$$\text{case } n_1 \textsf{ of } 0 \mapsto []; S \mapsto \lambda y_1. h_1 :: f_1\ t_1\ y_1 \sim (g_2\ h_2) :: (f_2\ t_2\ g_2) \mid \mathbf{r}_1 \sqsubseteq_{g_2} \mathbf{r}_2$$

Next, we apply CASE-L, which has the following two premises:

(A.3.i) $\Psi', m_1 = h_1 :: t_1, m_2 = h_2 :: t_2, n_1 = 0 \vdash [] \sim (g_2\ h_2) :: (f_2\ t_2\ g_2) \mid \mathbf{r}_1 \sqsubseteq_{g_2} \mathbf{r}_2$
(A.3.ii) $\Psi', m_1 = h_1 :: t_1, m_2 = h_2 :: t_2 \vdash \lambda y_1.h_1 :: f_1\ t_1\ y_1 \sim (g_2\ h_2) :: (f_2\ t_2\ g_2) \mid \forall y_1.n_1 = Sy_1 \Rightarrow (\mathbf{r}_1\ y_1) \sqsubseteq_{g_2} \mathbf{r}_2$

Premise (A.3.i) can be directly derived in HOL from the definition of $\sqsubseteq_{g_2}$. To prove (A.3.ii) we need to make use of our inductive hypothesis:

$$\forall k_1 k_2.(k_1,k_2) < (m_1,m_2) \Rightarrow k_1 = k_2 \Rightarrow \forall x_1 g_2.(f_1\ k_1\ x_1) \sqsubseteq_{g_2} (f_2\ k_2\ g_2)$$

In particular, from the premises $m_1 = h_1 :: t_1$ and $m_2 = h_2 :: t_2$ we can deduce $(t_1,t_2) < (m_1,m_2)$. Additionally, from the premise $m_1 = m_2$ we prove $t_1 = t_2$. Therefore, from the inductive hypothesis we derive $\forall x_1 g_2.(f_1\ t_1\ x_1) \sqsubseteq_{g_2} (f_2\ t_2\ g_2)$, and by definition of $\sqsubseteq_{g_2}$, and the fact that $h_1 = h_2$, for every y we can prove $h_1 :: (f_1\ t_1\ y) \sqsubseteq_{g_2} (g_2\ h_2) :: f_2\ t_2$. By Theorem 6, we can prove (A.3.ii).

We will now show how to prove (B) :

$$\begin{gathered}\Psi \vdash \mathit{map} \sim \mathit{take} \mid \forall m_1 m_2.m_1 \sqsubseteq_{g_2} m_2 \Rightarrow \\ (\forall g_1.g_1 = g_2 \Rightarrow \forall x_2.x_2 \geq |m_1| \Rightarrow (\mathbf{r}_1\ m_1\ g_1) \sqsubseteq (\mathbf{r}_2\ m_2\ x_2))\end{gathered}$$

On this branch we will also use LETREC. We have to prove a premise:

$$\Psi, \Phi \vdash \lambda g_1.e_2 \sim \lambda x_2.e_1 \mid \forall g_1.g_1 = g_2 \Rightarrow \forall x_2.x_2 \geq |m_1| \Rightarrow (\mathbf{r}_1\ g_1) \sqsubseteq (\mathbf{r}_2\ x_2)$$

where

$$\Phi \triangleq \left\{ \begin{array}{c} m_1 \sqsubseteq_{g_2} m_2, \\ \forall k_1 k_2.(k_1,k_2) < (m_1,m_2) \Rightarrow k_1 \sqsubseteq_{g_2} k_2 \Rightarrow \\ (\forall g_1.g_1 = g_2 \Rightarrow \forall x_2.x_2 \geq |k_1| \Rightarrow (\mathbf{r}_1\ k_1\ g_1) \sqsubseteq (\mathbf{r}_2\ k_2\ x_2)) \end{array} \right\}$$

We start by applying ABS. Our goal is to prove:

$$\Psi, \Phi, x_2 \geq |m_1|, g_1 = g_2 \vdash$$

$$\begin{array}{l} \text{case } m_1 \text{ of } [] \mapsto [] \\ ;\_ :: \_ \mapsto \lambda h_1 t_1.(g_1\ h_1) :: (f_1\ t_1\ g_1) \end{array} \sim \begin{array}{l} \text{case } m_2 \text{ of } [] \mapsto [] \\ ;\_ :: \_ \mapsto \lambda h_2 t_2.\text{case } x_2 \text{ of } 0 \mapsto [] \\ ;S \mapsto \lambda y_2.h_2 :: f_2\ t_2\ y_2 \end{array} \mid \mathbf{r}_1 \sqsubseteq \mathbf{r}_2$$

Notice that we have α-renamed the variables to have the appropriate subscript. Now we want to apply a CASE rule, but the lists over which we are matching are not necessarily of the same length. Therefore, we use the asynchronous LISTCASE-A rule. We have to prove four premises:

(B.1) $\Psi, \Phi, x_2 \geq |m_1|, g_1 = g_2, m_1 = [], m_2 = [] \vdash [] \sim [] \mid \mathbf{r}_1 \sqsubseteq \mathbf{r}_2$
(B.2) $\Psi, \Phi, x_2 \geq |m_1|, g_1 = g_2, m_1 = [] \vdash [] \sim \lambda h_2 t_2.\text{case } x_2 \text{ of } 0 \mapsto []; S \mapsto \lambda y_2.h_2 :: f_2\ t_2\ y_2 \mid \forall h_2 t_2.m_2 = h_2 :: t_2 \Rightarrow \mathbf{r}_1 \sqsubseteq (\mathbf{r}_2\ h_2\ t_2)$
(B.3) $\Psi, \Phi, x_2 \geq |m_1|, g_1 = g_2, m_2 = [] \vdash \lambda h_1 t_1.(g_1\ h_1) :: (f_1\ t_1\ g_1) \sim [] \mid \forall h_1 t_1.m_1 = h_1 :: t_1 \Rightarrow (\mathbf{r}_1\ h_1\ t_1) \sqsubseteq \mathbf{r}_2$
(B.4) $\Psi, \Phi, x_2 \geq |m_1|, g_1 = g_2 \vdash \lambda h_1 t_1.(g_1\ h_1) :: (f_1\ t_1\ g_1) \sim \lambda h_2 t_2.\text{case } x_2 \text{ of } 0 \mapsto []; S \mapsto \lambda y_2.h_2 :: f_2\ t_2\ y_2 \mid \forall h_1 t_1 h_2 t_2.m_1 = h_1 :: t_1 \Rightarrow m_2 = h_1 :: t_1 \Rightarrow (\mathbf{r}_1\ h_1\ t_1) \sqsubseteq (\mathbf{r}_2\ h_2\ t_2)$

Premises (B.1) and (B.2) are trivially derived from the definition of the $\sqsubseteq$ predicate. To prove premise (B.3) we see that we have premises $m_1 \sqsubseteq_{g_2} m_2$, $m_2 = []$, and $m_1 = h_1 :: t_2$, from which we can derive a contradiction.

It remains to prove (B.4). To do so, we apply ABS twice and then NATCASE-R, which has two premises:

(B.4.i) $\Psi, \Phi, x_2 \geq |m_1|, g_1 = g_2, m_1 = h_1 :: t_1, m_2 = h_1 :: t_1, x_2 = 0 \vdash (g_1\ h_1) :: (f_1\ t_1\ g_1) \sim [] \mid \mathbf{r}_1 \sqsubseteq \mathbf{r}_2$

(B.4.ii) $\Psi, \Phi, x_2 \geq |m_1|, g_1 = g_2, m_1 = h_1 :: t_1, m_2 = h_1 :: t_1 \vdash (g_1\ \ h_1) :: (f_1\ \ t_1\ \ g_1) \sim \lambda y_2.h_2 :: f_2\ t_2\ y_2 \mid$
$\forall y_2.x_2 = Sy_2 \Rightarrow \mathbf{r}_1 \sqsubseteq (\mathbf{r}_2\ y_2)$

To prove (B.4.i) we derive a contradiction between the premises. From $x_2 = 0$ and the premise $x_2 \geq |m_1|$ we derive $m_1 = []$ and, together with $m_1 = h_1 :: t_1$ we arrive at a contradiction by applying NC.

To prove (B.4.ii) we need to use the induction hypothesis. From $m_1 = h_1 :: t_1, m_2 = h_1 :: t_1$ we can prove that $|t_1| < |m_1|$ and $|t_2| < |m_2|$, so we can do a CUT with the i.h. and derive:

$$t_1 \sqsubseteq_{g_2} t_2 \Rightarrow (\forall g_1.g_1 = g_2 \Rightarrow \forall x_2.x_2 \geq |t_1| \Rightarrow (f_1\ t_1\ g_1) \sqsubseteq (f_2\ t_2\ x_2))$$

By assumption, $m_1 \sqsubseteq_{g_2} m_2$, so $t_1 \sqsubseteq_{g_2} t_2$. Moreover, also by assumption $g_1 = g_2$, and $Sy_2 = x_2 \geq |m_1| = S|t_1|$, so $y_2 \geq |t_1|$. So if we instantiate the i.h. with g_1 and y_2, and apply CUT again, we can prove:

$$(f_1\ t_1\ g_1) \sqsubseteq (f_2\ t_2\ y_2)$$

On the other hand, since $h_1 :: t_1 \sqsubseteq_{g_2} h_2 :: t_2$, then (by elimination of $\sqsubseteq_{g_2}$) we can derive $g_1 h_1 = h_2$ and by definition of $\sqsubseteq$, $(g_1\ h_1) :: (f_1\ t_1\ g_1) \sqsubseteq h_2 :: (f_2\ t_2\ y_2)$. So we can apply Theorem 6 and prove (B.4.ii). This ends the proof.

D.4 Proof of Theorem 24

We start by proving the key property of *restmin*, i.e.,

Lemma 38
Let $\mathsf{restmin}_1$ and $\mathsf{restmin}_2$ denote two α-renamings of $\mathsf{restmin}$ where every bound variable gets a subindex 1 or 2 respectively. Then,

$$\vdash \mathsf{restmin}_1 \sim \mathsf{restmin}_2 \mid \forall l_1 l_2.d(l_1, l_2) \leq \delta \Rightarrow \forall h_1 h_2.|h_1 - h_2| \leq \delta \Rightarrow D(\mathbf{r}_1\ l_1\ h_1, \mathbf{r}_2\ l_2\ h_2) \leq \delta$$

Proof
The proof is a simple synchronous derivation. We start by applying [LETREC], which gives us the inductive hypothesis:

$$\forall m_1 m_2.(|m_1|, |m_2|) \leq (|l_1|, |l_2|) \Rightarrow d(l_1, l_2) \leq \delta \Rightarrow \forall h_1 h_2.|h_1 - h_2| \leq \delta \Rightarrow D(\mathbf{r}_1\ l_1\ h_1, \mathbf{r}_2\ l_2\ h_2) \leq \delta$$

and we apply [ABS] immediately after. Then we do a synchronous case analysis with the [CASE] rule. In the empty list case, we simply instantiate the premise that $|a_1 - a_2| \leq \delta$. Since $d([], []) = 0 \leq \delta$, we can conclude that $D(([], a_1), ([], a_2)) \leq \delta$.

In the $h :: t$ case, we start by applying the [LET] rule three times. On the first two we need to use the lemma about max and min to prove that $|M_1 - M_2| \leq \delta$ and $|m_1 - m_2| \leq \delta$ and introduce them in the logical context.

On the third one, to introduce the hypothesis on $(rest, min)$, we instantiate the inductive hypothesis for $restmin$. Here we need to prove three facts: (1) $(t_1, t_2) \leq (l_1, l_2)$, (2) $|m_1 - m_2| \leq \delta$ and $d(t_1, t_2) \leq \delta$. Numbers (1) and (3) follow from the fact that $l_1 = h_1 :: t_1$ and $l_2 = h_2 :: t_2$. Number (2) follows from the let binding of m_1 and m_2. This introduces in the logical context the premises $d(rest_1, rest_2) \leq \delta$ and $|min_1 - min_2| \leq \delta$.

Finally, we need to show that $d(M_1 :: rest_1, M_1 :: rest_2) \leq \delta$. and that $|min_1 - min_2| \leq \delta$. The latter follows directly from the logical context, and the former follows from the inductive definition of the distance. $\square$

Now we need to prove for ssort' that:

$$\vdash \mathsf{ssort'}_1 \sim \mathsf{ssort'}_2 \mid \forall l_1 l_2 . d(l_1, l_2) \leq \delta \Rightarrow \forall n_1 n_2 . |l_1| = n_1 \wedge |l_2| = n_2 \leq \delta \Rightarrow d(\mathbf{r}_1\ l_1\ n_1, \mathbf{r}_2\ l_2\ n_2) \leq \delta$$

The derivation is entirely synchronous and routinary. The only interesting point is instantiating the lemma above for $\mathsf{restmin}$. This concludes the proof of the theorem.

D.5 Proof of Theorem 26

We need two straightforward lemmas in UHOL. The lemmas state that sorting preserves the length and minimum element of a list.

Lemma 39
Let $\tau \triangleq \mathsf{list}_{\mathbb{N}} \to \mathsf{list}_{\mathbb{N}}$. Then, (1) $\bullet \mid \bullet \vdash \mathsf{insert} : \mathbb{N} \to \tau \mid \forall x\, l . |\pi_1(\mathbf{r}\ x\ l)| = 1 + |l|$, and (2) $\bullet \mid \bullet \vdash \mathsf{isort} : \tau \mid \forall x . |\pi_1(\mathbf{r}\ x)| = |x|$.

Lemma 40
Let $\tau \triangleq \mathsf{list}_{\mathbb{N}} \to \mathsf{list}_{\mathbb{N}}$. Then, (1) $\bullet \mid \bullet \vdash \mathsf{insert} : \mathbb{N} \to \tau \mid \forall x\, l . \mathsf{lmin}(\pi_1(\mathbf{r}\ x\ l)) = \mathsf{min}(x, \mathsf{lmin}(l))$, and (2) $\bullet \mid \bullet \vdash \mathsf{isort} : \tau \mid \forall x . \mathsf{lmin}(\pi_1(\mathbf{r}\ x)) = \mathsf{lmin}(x)$.

Proof of Theorem 26
We prove the theorem using LETREC. We actually show the following stronger theorem, which yields a stronger induction hypothesis in the proof.

$$\begin{gathered}\bullet \mid \bullet \vdash \mathsf{isort} : \tau \sim \mathsf{isort} : \tau \mid \forall x_1\, x_2 . (\mathsf{sorted}(x_1) \wedge |x_1| = |x_2|) \Rightarrow \\ (\pi_2(\mathbf{r}_1\ x_1) \leq \pi_2(\mathbf{r}_2\ x_2)) \wedge \underline{(\mathbf{r}_1\ x_1 = \mathsf{isort}\ x_1) \wedge (\mathbf{r}_2\ x_2 = \mathsf{isort}\ x_2)}\end{gathered}$$

Let ι denote the inductive hypothesis:

$$\begin{aligned}\iota \triangleq \forall m_1\, m_2 . (|m_1|, |m_2|) < (|x_1|, |x_2|) \Rightarrow\ & (\mathsf{sorted}(m_1) \wedge |m_1| = |m_2|) \\ & \Rightarrow \pi_2(isort_1\ m_1) \leq \pi_2(isort_2\ m_2) \wedge \\ & (isort_1\ m_1 = \mathsf{isort}\ m_1) \wedge (isort_2\ m_2 = \mathsf{isort}\ m_2)\end{aligned}$$

and e denote the body of the function isort:

$$\begin{aligned} e \triangleq \text{case } l \text{ of } & [] \mapsto ([], 0); \\ & \_ :: \_ \mapsto \lambda h\, t.\ \text{let } s = isort\ t \\ & \qquad\qquad\quad \text{let } s' = \mathsf{insert}\ h\ (\pi_1\ s) \text{ in} \\ & \qquad\qquad\quad (\pi_1\ s', (\pi_2\ s) + (\pi_2\ s')) \end{aligned}$$

By LETREC, it suffices to prove the following (we omit simple types for easier reading; they play no essential role in the proof).

$$\begin{array}{c} isort_1, isort_2, x_1, x_2 \mid \mathsf{sorted}(x_1), |x_1| = |x_2|, \iota \vdash \\ e[isort_1/isort][x_1/l] \sim e[isort_2/isort][x_2/l] \mid \left(\begin{array}{l} \pi_2\ \mathbf{r}_1 \le \pi_2\ \mathbf{r}_2 \\ \wedge\, \mathbf{r}_1 = \mathsf{isort}\ x_1 \\ \wedge\, \mathbf{r}_2 = \mathsf{isort}\ x_2 \end{array} \right) \end{array}$$

Following the structure of e, we next apply the rule LISTCASE. This yields the following two main proof obligations, corresponding to the two case branches (the third proof obligation, $x_1 = [] \Leftrightarrow x_2 = []$ follows immediately from the assumption $|x_1| = |x_2|$).

$$\begin{array}{c} isort_1, isort_2, x_1, x_2 \mid \mathsf{sorted}(x_1), |x_1| = |x_2|, \iota, x_1 = x_2 = [] \vdash ([], 0) \sim ([], 0) \mid \\ (\pi_2\ \mathbf{r}_1 \le \pi_2\ \mathbf{r}_2) \wedge (\mathbf{r}_1 = \mathsf{isort}\ x_1) \wedge (\mathbf{r}_2 = \mathsf{isort}\ x_2) \end{array} \tag{D 1}$$

$$\begin{array}{c} isort_1, isort_2, x_1, x_2, h_1, t_1, h_2, t_2 \mid \mathsf{sorted}(x_1), |x_1| = |x_2|, \iota, \underline{x_1 = h_1 :: t_1, x_2 = h_2 :: t_2} \vdash \\ \begin{array}{l} \text{let } s = isort_1\ t_1 \\ \text{let } s' = \mathsf{insert}\ h_1\ (\pi_1\ s) \text{ in} \\ (\pi_1\ s', (\pi_2\ s) + (\pi_2\ s')) \end{array} \sim \begin{array}{l} \text{let } s = isort_2\ t_2 \\ \text{let } s' = \mathsf{insert}\ h_2\ (\pi_1\ s) \text{ in} \\ (\pi_1\ s', (\pi_2\ s) + (\pi_2\ s')) \end{array} \left| \begin{array}{l} \pi_2\ \mathbf{r}_1 \le \pi_2\ \mathbf{r}_2 \\ \wedge\, \mathbf{r}_1 = \mathsf{isort}\ x_1 \\ \wedge\, \mathbf{r}_2 = \mathsf{isort}\ x_2 \end{array} \right. \end{array} \tag{D 2}$$

(D 1) is immediate: By Theorem 6, it suffices to show that $(\pi_2([], 0) \le \pi_2([], 0)) \wedge (([], 0) = \mathsf{isort}\ x_1) \wedge (([], 0) = \mathsf{isort}\ x_2)$. Since $x_1 = x_2 = []$ by assumption here, this is equivalent to $(\pi_2([], 0) \le \pi_2([], 0)) \wedge (([], 0) = \mathsf{isort}\ []) \wedge (([], 0) = \mathsf{isort}\ [])$, which is trivial by direct computation.

To prove (D 2), we expand the outermost occurrences of let in both to function applications using the definition let $x = e_1$ in $e_2 \triangleq (\lambda x.e_2)\ e_1$. Applying the rules APP and ABS, it suffices to prove the following for any ϕ of our choice.

$$\begin{array}{c} isort_1, isort_2, x_1, x_2, h_1, t_1, h_2, t_2 \mid \mathsf{sorted}(x_1), |x_1| = |x_2|, \iota, x_1 = h_1 :: t_1, x_2 = h_2 :: t_2 \vdash \\ isort_1\ t_1 \sim isort_2\ t_2 \mid \phi \end{array} \tag{D 3}$$

$$\begin{array}{l} isort_1, isort_2, x_1, x_2, h_1, t_1, h_2, t_2, s_1, s_2 \mid \\ \mathsf{sorted}(x_1), |x_1| = |x_2|, \iota, x_1 = h_1 :: t_1, x_2 = h_2 :: t_2 \underline{\phi[s_1/\mathbf{r}_1][s_2/\mathbf{r}_2]} \vdash \\ \begin{array}{l} \text{let } s' = \mathsf{insert}\ h_1\ (\pi_1\ s_1) \text{ in} \\ (\pi_1\ s', (\pi_2\ s_1) + (\pi_2\ s')) \end{array} \sim \begin{array}{l} \text{let } s' = \mathsf{insert}\ h_2\ (\pi_1\ s_2) \text{ in} \\ (\pi_1\ s', (\pi_2\ s_2) + (\pi_2\ s')) \end{array} \left| \begin{array}{l} \pi_2\ \mathbf{r}_1 \le \pi_2\ \mathbf{r}_2 \\ \wedge\, \mathbf{r}_1 = \mathsf{isort}\ x_1 \\ \wedge\, \mathbf{r}_2 = \mathsf{isort}\ x_2 \end{array} \right. \end{array} \tag{D 4}$$

We choose ϕ as follows:

$$\phi \triangleq \pi_2\ \mathbf{r}_1 \le \pi_2\ \mathbf{r}_2 \wedge \mathbf{r}_1 = \mathsf{isort}(t_1) \wedge \mathbf{r}_2 = \mathsf{isort}(t_2) \wedge |\pi_1\ \mathbf{r}_1| = |\pi_1\ \mathbf{r}_2| \wedge \mathsf{lmin}(t_1) = \mathsf{lmin}(\pi_1\ \mathbf{r}_1)$$

Proof of (D 3): By Theorem 6, it suffices to prove the following five statements in HOL under the context of (D 3). These statements correspond to the five conjuncts of ϕ.

$$\pi_2(isort_1\ t_1) \le \pi_2(isort_2\ t_2) \tag{D 5}$$

$$isort_1\ t_1 = \mathsf{isort}\ t_1 \tag{D 6}$$

$$isort_1\ t_2 = \mathsf{isort}\ t_2 \tag{D 7}$$

$$|\pi_1(isort_1\ t_1)| = |\pi_1(isort_2\ t_2)| \tag{D 8}$$

$$\mathsf{lmin}(t_1) = \mathsf{lmin}(\pi_1(isort_1\ t_1)) \tag{D 9}$$

(D 5)–(D 7) follow from the induction hypothesis ι instantiated with $m_1 := t_1, m_2 := t_2$. Note that because $x_1 = h_1 :: t_1$ and $x_2 = h_2 :: t_2$, we can prove (in HOL) that $(|t_1|,|t_2|) < (|x_1|,|x_2|)$. Since, $|x_1| = |x_2|$, $x_1 = h_1 :: t_1$ and $x_2 = h_2 :: t_2$, we can also prove that $|t_1| = |t_2|$. Finally, from the axiomatic definition of sorted and the assumption $\mathsf{sorted}(x_1)$ it follows that $\mathsf{sorted}(t_1)$. These together allow us to instantiate the i.h. ι and immediately derive (D 5)–(D 7).

To prove (D 8), we use (D 6) and (D 7), which reduces (D 8) to $|\pi_1(\mathsf{isort}\ t_1)| = |\pi_1(\mathsf{isort}\ t_2)|$. To prove this, we apply Theorem 3 to Lemma 39, yielding $\forall x. |\pi_1(\mathsf{isort}\ x)| = |x|$. Hence, we can further reduce our goal to proving $|t_1| = |t_2|$, which we already did above.

To prove (D 9), we use (D 6), which reduces (D 9) to $\mathsf{lmin}(t_1) = \mathsf{lmin}(\pi_1(\mathsf{isort}\ t_1))$. This follows immediately from Theorem 3 applied to Lemma 40.
This proves (D 3).

Proof of (D 4): We expand the definition of let and apply the rules APP and ABS to reduce (D 4) to proving the following for any ϕ'.

$$\begin{array}{l} isort_1, isort_2, x_1, x_2, h_1, t_1, h_2, t_2, s_1, s_2 \mid \\ \mathsf{sorted}(x_1), |x_1| = |x_2|, \iota, x_1 = h_1 :: t_1, x_2 = h_2 :: t_2, \phi[s_1/\mathbf{r}_1][s_2/\mathbf{r}_2] \vdash \\ \mathsf{insert}\ h_1\ (\pi_1\ s_1) \sim \mathsf{insert}\ h_2\ (\pi_1\ s_2) \mid \phi' \end{array} \tag{D 10}$$

$$\begin{array}{l} isort_1, isort_2, x_1, x_2, h_1, t_1, h_2, t_2, s_1, s_2, s'_1, s'_2 \mid \\ \mathsf{sorted}(x_1), |x_1| = |x_2|, \iota, x_1 = h_1 :: t_1, x_2 = h_2 :: t_2 \phi[s_1/\mathbf{r}_1][s_2/\mathbf{r}_2], \phi'[s'_1/\mathbf{r}_1][s'_2/\mathbf{r}_2] \vdash \\ (\pi_1\ s'_1, (\pi_2\ s_1) + (\pi_2\ s'_1)) \sim (\pi_1\ s'_2, (\pi_2\ s_2) + (\pi_2\ s'_2)) \left| \begin{array}{l} \pi_2\ \mathbf{r}_1 \le \pi_2\ \mathbf{r}_2 \\ \wedge\ \mathbf{r}_1 = \mathsf{isort}\ x_1 \\ \wedge\ \mathbf{r}_2 = \mathsf{isort}\ x_2 \end{array} \right. \end{array} \tag{D 11}$$

We pick the following ϕ':

$$\phi' \triangleq \pi_2\ \mathbf{r}_1 \leq \pi_2\ \mathbf{r}_2 \wedge \mathbf{r}_1 = \mathsf{insert}\ h_1\ (\pi_1\ s_1) \wedge \mathbf{r}_2 = \mathsf{insert}\ h_2\ (\pi_1\ s_2)$$

Proof of (D 10): We start by applying Theorem 6. This yields three subgoals in HOL, corresponding to the three conjuncts in ϕ':

$$\pi_2(\mathsf{insert}\ h_1\ (\pi_1\ s_1)) \leq \pi_2(\mathsf{insert}\ h_2\ (\pi_1\ s_2)) \tag{D 12}$$

$$\mathsf{insert}\ h_1\ (\pi_1\ s_1) = \mathsf{insert}\ h_1\ (\pi_1\ s_1) \tag{D 13}$$

$$\mathsf{insert}\ h_2\ (\pi_1\ s_2) = \mathsf{insert}\ h_2\ (\pi_1\ s_2) \tag{D 14}$$

(D 13) and (D 14) are trivial, so we only have to prove (D 12). Since $s_1 = \mathsf{isort}\ t_1$ and $s_2 = \mathsf{isort}\ t_2$ are conjuncts in the assumption $\phi[s_1/\mathbf{r}_1][s_2/\mathbf{r}_2]$, (D 12) is equivalent to:

$$\pi_2(\mathsf{insert}\ h_1\ (\pi_1(\mathsf{isort}\ t_1))) \leq \pi_2(\mathsf{insert}\ h_2\ (\pi_1(\mathsf{isort}\ t_2))) \tag{D 15}$$

To prove this, we split cases on the shapes of $\pi_1(\mathsf{isort}\ t_1)$ and $\pi_1(\mathsf{isort}\ t_2)$. From the conjuncts in $\phi[s_1/\mathbf{r}_1][s_2/\mathbf{r}_2]$, it follows immediately that $|\pi_1(\mathsf{isort}\ t_1)| = |\pi_1(\mathsf{isort}\ t_2)|$. Hence, only two cases apply:

Case: $\pi_1(\mathsf{isort}\ t_1) = \pi_1(\mathsf{isort}\ t_2) = []$. In this case, by direct computation, $\pi_2(\mathsf{insert}\ h_1\ (\pi_1(\mathsf{isort}\ t_1))) = \pi_2(\mathsf{insert}\ h_1\ []) = \pi_2([h_1], 0) = 0$. Similarly, and $\pi_2(\mathsf{insert}\ h_2\ (\pi_1(\mathsf{isort}\ t_2))) = 0$. So, the result follows trivially.

Case: $\pi_1(\mathsf{isort}\ t_1) = h_1' :: t_1'$ and $\pi_1(\mathsf{isort}\ t_2) = h_2' :: t_2'$. We first argue that $h_1 \leq h_1'$. Note that from the second and fifth conjuncts in $\phi[s_1/\mathbf{r}_1][s_2/\mathbf{r}_2]$, it follows that $\mathsf{lmin}(t_1) = \mathsf{lmin}(\pi_1(\mathsf{isort}\ t_1))$. Since $\pi_1(\mathsf{isort}\ t_1) = h_1' :: t_1'$, we further get $\mathsf{lmin}(t_1) = \mathsf{lmin}(\pi_1(\mathsf{isort}\ t_1)) = \mathsf{lmin}(h_1' :: t_1') = \mathsf{min}(h_1', \mathsf{lmin}(t_1')) \leq h_1'$. Finally, from the axiomatic definition of $\mathsf{sorted}(x_1)$ and $x_1 = h_1 :: t_1$, we derive $h_1 \leq \mathsf{lmin}(t_1)$. Combining, we get $h_1 \leq \mathsf{lmin}(t_1) \leq h_1'$.

Next, $\pi_2(\mathsf{insert}\ h_1\ (\pi_1(\mathsf{isort}\ t_1))) = \pi_2(\mathsf{insert}\ h_1\ (h_1' :: t_1'))$. Expanding the definition of insert and using $h_1 \leq h_1'$, we immediately get $\pi_2(\mathsf{insert}\ h_1\ (\pi_1(\mathsf{isort}\ t_1))) = \pi_2(\mathsf{insert}\ h_1\ (h_1' :: t_1')) = \pi_2(h_1 :: h_1' :: t_1', 1) = 1$. On the other hand, it is fairly easy to prove (by case analyzing the result of $h_2 \leq h_2'$) that $\pi_2(\mathsf{insert}\ h_2\ (\pi_1(\mathsf{isort}\ t_2))) = \pi_2(\mathsf{insert}\ h_2\ (h_2' :: t_2')) \geq 1$. Hence, $\pi_2(\mathsf{insert}\ h_1\ (\pi_1(\mathsf{isort}\ t_1))) = 1 \leq \pi_2(\mathsf{insert}\ h_2\ (\pi_1(\mathsf{isort}\ t_2)))$. This proves (D 15) and, hence, (D 12) and (D 10).

Proof of (D 11): By Theorem 6, it suffices to show the following in HOL, under the assumptions of (D 11):

$$\pi_2(\pi_1\ s_1', (\pi_2\ s_1) + (\pi_2\ s_1')) \leq \pi_2(\pi_1\ s_2', (\pi_2\ s_2) + (\pi_2\ s_2')) \tag{D 16}$$

$$(\pi_1\ s_1', (\pi_2\ s_1) + (\pi_2\ s_1')) = \mathsf{isort}\ x_1 \tag{D 17}$$

$$(\pi_1\ s_2', (\pi_2\ s_2) + (\pi_2\ s_2')) = \mathsf{isort}\ x_2 \tag{D 18}$$

By computation, (D 16) is equivalent to $(\pi_2\ s_1) + (\pi_2\ s_1') \leq (\pi_2\ s_2) + (\pi_2\ s_2')$. Using the definition of ϕ, it is easy to see that $\pi_2\ s_1 \leq \pi_2\ s_2$ is a conjunct in the assumption $\phi[s_1/\mathbf{r}_1][s_2/\mathbf{r}_2]$. Similarly, using the definition of ϕ', $\pi_2\ s_1' \leq \pi_2\ s_2'$ is a conjunct in the assumption $\phi'[s_1'/\mathbf{r}_1][s_2'/\mathbf{r}_2]$. (D 16) follows immediately from these.

To prove (D 17), note that since $x_1 = h_1 :: t_1$, expanding the definition of isort, we get

$$\mathsf{isort}\ x_1 = (\pi_1(\mathsf{insert}\ h_1\ (\pi_1(\mathsf{isort}\ t_1))), \pi_2(\mathsf{isort}\ t_1) + \pi_2(\mathsf{insert}\ h_1\ (\pi_1(\mathsf{isort}\ t_1))))$$

Matching with the left side of (D 17), it suffices to show that $s_1' = \mathsf{insert}\ h_1\ (\pi_1(\mathsf{isort}\ t_1))$ and $s_1 = \mathsf{isort}\ t_1$. These are immediate: $s_1 = \mathsf{isort}\ t_1$ is a conjunct in the assumption $\phi[s_1/\mathbf{r}_1][s_2/\mathbf{r}_2]$, while $s_1' = \mathsf{insert}\ h_1\ (\pi_1(\mathsf{isort}\ t_1))$ follows trivially from this and the conjunct $s_1' = \mathsf{insert}\ h_1\ (\pi_1\ s_1)$ in $\phi'[s_1'/\mathbf{r}_1][s_2'/\mathbf{r}_2]$. This proves (D 17).

The proof of (D 18) is similar to that of (D 17).

This proves (D 11) and, hence, (D 4). $\square$

E Full RHOL rules

The full set of RHOL rules is in the following figures:

$$\frac{\Gamma, x_1 : \tau_1, x_2 : \tau_2 \mid \Psi, \phi' \vdash t_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi \quad x_1 \notin \mathsf{FV}(t_2), x_2 \notin \mathsf{FV}(t_1)}{\Gamma \mid \Psi \vdash \lambda x_1.t_1 : \tau_1 \to \sigma_1 \sim \lambda x_2.t_2 : \tau_2 \to \sigma_2 \mid \forall x_1, x_2.\phi' \Rightarrow \phi[\mathbf{r}_1\ x_1/\mathbf{r}_1][\mathbf{r}_2\ x_2/\mathbf{r}_2]}\ \mathsf{ABS}$$

$$\frac{\begin{array}{c}\Gamma \mid \Psi \vdash t_1 : \tau_1 \to \sigma_1 \sim t_2 : \tau_2 \to \sigma_2 \mid \forall x_1, x_2.\phi'[x_1/\mathbf{r}_1][x_2/\mathbf{r}_2] \Rightarrow \phi[\mathbf{r}_1\ x_1/\mathbf{r}_1][\mathbf{r}_2\ x_2/\mathbf{r}_2]\\ \Gamma \mid \Psi \vdash u_1 : \tau_1 \sim u_2 : \tau_2 \mid \phi'\end{array}}{\Gamma \mid \Psi \vdash t_1 u_1 : \sigma_1 \sim t_2 u_2 : \sigma_2 \mid \phi[u_1/x_1][u_2/x_2]}\ \mathsf{APP}$$

$$\frac{\Gamma \mid \Psi \vdash_{\mathsf{HOL}} \phi[0/\mathbf{r}_1][0/\mathbf{r}_2]}{\Gamma \mid \Psi \vdash 0 : \mathbb{N} \sim 0 : \mathbb{N} \mid \phi}\ \mathsf{ZERO}$$

$$\frac{\begin{array}{c}\Gamma \mid \Psi \vdash t_1 : \mathbb{N} \sim t_2 : \mathbb{N} \mid \phi'\\ \Gamma \mid \Psi \vdash_{\mathsf{HOL}} \forall x_1 x_2 \phi'[x_1/\mathbf{r}_1][x_2/\mathbf{r}_2] \Rightarrow \phi[S x_1/\mathbf{r}_1][S x_2/\mathbf{r}_2]\end{array}}{\Gamma \mid \Psi \vdash S t_1 : \mathbb{N} \sim S t_2 : \mathbb{N} \mid \phi}\ \mathsf{SUCC}$$

$$\frac{\Gamma \mid \Psi \vdash \phi[x_1/\mathbf{r}_1][x_2/\mathbf{r}_2] \quad \Gamma \vdash x_1 : \sigma_1 \quad \Gamma \vdash x_1 : \sigma_1}{\Gamma \mid \Psi \vdash x_1 : \sigma_1 \sim x_2 : \sigma_2 \mid \phi}\ \mathsf{VAR}$$

$$\frac{\Gamma \mid \Psi \vdash t_1 : \tau_1 \sim t_2 : \tau_2 \mid \phi' \quad \Gamma, x_1 : \tau_1, x_2 : \tau_2 \mid \Psi, \phi'[x_1/\mathbf{r}_1][x_2/\mathbf{r}_2] \vdash u_1 : \sigma_1 \sim u_2 : \sigma_2 \mid \phi}{\Gamma \mid \Psi \vdash \text{let } x_1 = t_1 \text{ in } u_1 : \sigma_1 \sim \text{let } x_2 = t_2 \text{ in } u_2 : \sigma_2 \mid \phi}\ \mathsf{LET}$$

$$\frac{\Gamma \mid \Psi \vdash_{\mathsf{HOL}} \phi[\mathsf{tt}/\mathbf{r}_1][\mathsf{tt}/\mathbf{r}_2]}{\Gamma \mid \Psi \vdash \mathsf{tt} : \mathbb{B} \sim \mathsf{tt} : \mathbb{B} \mid \phi}\ \mathsf{TRUE}$$

$$\frac{\Gamma \mid \Psi \vdash_{\mathsf{HOL}} \phi[\mathsf{ff}/\mathbf{r}_1][\mathsf{ff}/\mathbf{r}_2]}{\Gamma \mid \Psi \vdash \mathsf{ff} : \mathbb{B} \sim \mathsf{ff} : \mathbb{B} \mid \phi}\ \mathsf{FALSE}$$

$$\frac{\Gamma \mid \Psi \vdash_{\mathsf{HOL}} \phi[[]/\mathbf{r}_1][[]/\mathbf{r}_2]}{\Gamma \mid \Psi \vdash [] : \mathsf{list}_{\sigma_1} \sim [] : \mathsf{list}_{\sigma_2} \mid \phi}\ \mathsf{NIL}$$

$$\frac{\begin{array}{c}\Gamma \mid \Psi \vdash h_1 : \sigma_1 \sim h_2 : \sigma_2 \mid \phi' \qquad \Gamma \mid \Psi \vdash t_1 : \mathsf{list}_{\sigma_1} \sim t_2 : \mathsf{list}_{\sigma_2} \mid \phi''\\ \Gamma \mid \Psi \vdash_{\mathsf{HOL}} \forall x_1 x_2 y_1 y_2.\phi'[x_1/\mathbf{r}_1][x_2/\mathbf{r}_2] \Rightarrow \phi''[y_1/\mathbf{r}_1][y_2/\mathbf{r}_2] \Rightarrow \phi[x_1 :: y_1/\mathbf{r}_1][x_2 :: y_2/\mathbf{r}_2]\end{array}}{\Gamma \mid \Psi \vdash h_1 :: t_1 : \mathsf{list}_{\sigma_1} \sim h_2 :: t_2 : \mathsf{list}_{\sigma_2} \mid \phi}\ \mathsf{CONS}$$

$$\frac{\begin{array}{c}\Gamma \mid \Psi \vdash t_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi' \qquad \Gamma \mid \Psi \vdash u_1 : \tau_1 \sim u_2 : \tau_2 \mid \phi''\\ \Gamma \mid \Psi \vdash_{\mathsf{HOL}} \forall x_1 x_2 y_1 y_2.\phi'[x_1/\mathbf{r}_1][x_2/\mathbf{r}_2] \Rightarrow \phi''[y_1/\mathbf{r}_1][y_2/\mathbf{r}_2] \Rightarrow \phi[\langle x_1, y_1\rangle/\mathbf{r}_1][\langle x_2, y_2\rangle/\mathbf{r}_2]\end{array}}{\Gamma \mid \Psi \vdash \langle t_1, u_1\rangle : \sigma_1 \times \tau_1 \sim \langle t_2, u_2\rangle : \sigma_2 \times \tau_2 \mid \phi}\ \mathsf{PAIR}$$

$$\frac{\Gamma \mid \Psi \vdash t_1 : \sigma_1 \times \tau_1 \sim t_2 : \sigma_2 \times \tau_2 \mid \phi[\pi_i(\mathbf{r}_1)/\mathbf{r}_1][\pi_i(\mathbf{r}_2)/\mathbf{r}_2]}{\Gamma \mid \Psi \vdash \pi_i(t_1) : \sigma_1 \sim \pi_i(t_2) : \sigma_2 \mid \phi}\ \mathsf{PROJ_i}$$

Fig. E 1. Core two-sided rules

$$\frac{\Gamma \mid \Psi \vdash t_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi' \quad \Gamma \mid \Psi \vdash_{\mathsf{HOL}} \phi'[t_1/\mathbf{r}_1][t_2/\mathbf{r}_2] \Rightarrow \phi[t_1/\mathbf{r}_1][t_2/\mathbf{r}_2]}{\Gamma \mid \Psi \vdash t_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi} \ \mathsf{SUB}$$

$$\frac{\Gamma \mid \Psi' \vdash t_1 : \sigma_2 \sim t_2 : \sigma_2 \mid \phi \quad \Gamma \mid \Psi' \vdash t_1 : \sigma_2 \sim t_2 : \sigma_2 \mid \phi'}{\Gamma \mid \Psi' \vdash t_1 : \sigma_2 \sim t_2 : \sigma_2 \mid \phi \wedge \phi'} \ \wedge_{\mathsf{I}}$$

$$\frac{\Gamma \mid \Psi', \phi'[t_1/\mathbf{r}_1][t_2/\mathbf{r}_2] \vdash t_1 : \sigma_2 \sim t_2 : \sigma_2 \mid \phi}{\Gamma \mid \Psi' \vdash t_1 : \sigma_2 \sim t_2 : \sigma_2 \mid \phi' \Rightarrow \phi} \ \Rightarrow_{\mathsf{I}}$$

$$\frac{\Gamma \mid \Psi \vdash t_1 : \sigma_1 \mid \phi[\mathbf{r}/\mathbf{r}_1][t_2/\mathbf{r}_2]}{\Gamma \mid \Psi \vdash t_1 : \sigma_1 \sim t_2 : \sigma_1 \mid \phi} \ \mathsf{UHOL-L}$$

Fig. E 2. Structural rules

$$\dfrac{\Gamma, x_1 : \tau_1 \mid \Psi, \phi' \vdash t_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi \quad x_1 \notin \mathsf{FV}(t_2)}{\Gamma \mid \Psi \vdash \lambda x_1.t_1 : \tau_1 \to \sigma_1 \sim t_2 : \sigma_2 \mid \forall x_1.\phi' \Rightarrow \phi[\mathbf{r}_1\ x_1/\mathbf{r}_1]}\ \mathsf{ABS{-}L}$$

$$\dfrac{\begin{array}{c}\Gamma \mid \Psi \vdash t_1 : \tau_1 \to \sigma_1 \sim u_2 : \sigma_2 \mid \forall x_1.\phi'[x_1/\mathbf{r}_1] \Rightarrow \phi[\mathbf{r}_1\ x_1/\mathbf{r}_1] \\ \Gamma \mid \Psi \vdash u_1 : \sigma_1 \mid \phi'\end{array}}{\Gamma \mid \Psi \vdash t_1 u_1 : \sigma_1 \sim u_2 : \sigma_2 \mid \phi[u_1/x_1]}\ \mathsf{APP{-}L}$$

$$\dfrac{\begin{array}{c}\Gamma \vdash t_2 : \sigma_2 \\ \Gamma \mid \Psi \vdash_{\mathsf{HOL}} \phi[0/\mathbf{r}_1][t_2/\mathbf{r}_2]\end{array}}{\Gamma \mid \Psi \vdash 0 : \mathbb{N} \sim t_2 : \sigma_2 \mid \phi}\ \mathsf{ZERO{-}L}$$

$$\dfrac{\begin{array}{c}\Gamma \mid \Psi \vdash t_1 : \mathbb{N} \sim t_2 : \sigma_2 \mid \phi' \\ \Gamma \mid \Psi \vdash_{\mathsf{HOL}} \forall x_1 x_2 \phi'[x_1/\mathbf{r}_1][x_2/\mathbf{r}_2] \Rightarrow \phi[Sx_1/\mathbf{r}_1][x_2/\mathbf{r}_2]\end{array}}{\Gamma \mid \Psi \vdash St_1 : \mathbb{N} \sim t_2 : \sigma_2 \mid \phi}\ \mathsf{SUCC{-}L}$$

$$\dfrac{\Gamma \mid \Psi \vdash_{\mathsf{HOL}} \phi[\mathsf{tt}/\mathbf{r}_1][t_2/\mathbf{r}_2] \quad \Gamma \vdash t_2 : \sigma_2}{\Gamma \mid \Psi \vdash \mathsf{tt} : \mathbb{B} \sim t_2 : \sigma_2 \mid \phi}\ \mathsf{TRUE-L}$$

$$\dfrac{\Gamma \mid \Psi \vdash_{\mathsf{HOL}} \phi[\mathsf{ff}/\mathbf{r}_1][t_2/\mathbf{r}_2] \quad \Gamma \vdash t_2 : \sigma_2}{\Gamma \mid \Psi \vdash \mathsf{ff} : \mathbb{B} \sim t_2 : \sigma_2 \mid \phi}\ \mathsf{FALSE-L}$$

$$\dfrac{\phi[x_1/\mathbf{r}_1] \in \Psi \quad \mathbf{r}_2 \notin FV(\phi) \quad \Gamma \vdash t_2 : \sigma_2}{\Gamma \mid \Psi \vdash x_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi}\ \mathsf{VAR{-}L}$$

$$\dfrac{\Gamma \mid \Psi \vdash \phi[[]/\mathbf{r}_1][t_2/\mathbf{r}_2] \quad \Gamma \vdash t_2 : \sigma_2}{\Gamma \mid \Psi \vdash [] : \mathsf{list}_{\sigma_1} \sim t_2 : \sigma_2 \mid \phi}\ \mathsf{NIL{-}L}$$

$$\dfrac{\begin{array}{c}\Gamma \mid \Psi \vdash h_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi' \qquad \Gamma \mid \Psi \vdash t_1 : \mathsf{list}_{\sigma_1} \sim t_2 : \sigma_2 \mid \phi'' \\ \Gamma \mid \Psi \vdash_{\mathsf{HOL}} \forall x_1 x_2 y_1.\phi'[x_1/\mathbf{r}_1][x_2/\mathbf{r}_2] \Rightarrow \phi''[y_1/\mathbf{r}_1][x_2/\mathbf{r}_2] \Rightarrow \phi[x_1 :: y_1/\mathbf{r}_1][x_2/\mathbf{r}_2]\end{array}}{\Gamma \mid \Psi \vdash h_1 :: t_1 : \mathsf{list}_{\sigma_1} \sim t_2 : \sigma_2 \mid \phi}\ \mathsf{CONS{-}L}$$

$$\dfrac{\begin{array}{c}\Gamma \mid \Psi \vdash t_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi' \qquad \Gamma \mid \Psi \vdash u_1 : \tau_1 \sim t_2 : \sigma_2 \mid \phi'' \\ \Gamma \mid \Psi \vdash_{\mathsf{HOL}} \forall x_1 x_2 y_1.\phi'[x_1/\mathbf{r}_1][x_2/\mathbf{r}_2] \Rightarrow \phi''[y_1/\mathbf{r}_1][x_2/\mathbf{r}_2] \Rightarrow \phi[\langle x_1, y_1\rangle/\mathbf{r}_1][x_2/\mathbf{r}_2]\end{array}}{\Gamma \mid \Psi \vdash \langle t_1, u_1\rangle : \sigma_1 \times \tau_1 \sim t_2 : \sigma_2 \mid \phi}\ \mathsf{PAIR{-}L}$$

$$\dfrac{\Gamma \mid \Psi \vdash t_1 : \sigma_1 \times \tau_1 \sim t_2 : \sigma_2 \mid \phi[\pi_1(\mathbf{r}_1)/\mathbf{r}_1]}{\Gamma \mid \Psi \vdash \pi_1(t_1) : \sigma_1 \sim t_2 : \sigma_2 \mid \phi}\ \mathsf{PROJ_1{-}L}$$

Fig. E 3. Core one-sided rules

$$\frac{\begin{array}{c}\Gamma \mid \Psi \vdash t_1 : \mathbb{B} \sim t_2 : \mathbb{B} \mid (\mathbf{r}_1 = \mathsf{tt} \wedge \mathbf{r}_2 = \mathsf{tt}) \vee (\mathbf{r}_1 = \mathsf{ff} \wedge \mathbf{r}_2 = \mathsf{ff}) \\ \Gamma \mid \Psi, t_1 = \mathsf{tt}, t_2 = \mathsf{tt} \vdash u_1 : \sigma_1 \sim u_2 : \sigma_2 \mid \phi \\ \Gamma \mid \Psi, t_1 = \mathsf{ff}, t_2 = \mathsf{ff} \vdash v_1 : \sigma_1 \sim v_2 : \sigma_2 \mid \phi\end{array}}{\Gamma \mid \Psi \vdash \text{case } t_1 \text{ of } \mathsf{tt} \mapsto u_1; \mathsf{ff} \mapsto v_1 : \sigma_1 \sim \text{case } t_2 \text{ of } \mathsf{tt} \mapsto u_2; \mathsf{ff} \mapsto v_2 : \sigma_2 \mid \phi}\ \mathsf{BOOLCASE}$$

$$\frac{\begin{array}{c}\Gamma \mid \Psi \vdash t_1 : \mathbb{N} \sim t_2 : \mathbb{N} \mid \mathbf{r}_1 = 0 \Leftrightarrow \mathbf{r}_2 = 0 \\ \Gamma \mid \Psi, t_1 = 0, t_2 = 0 \vdash u_1 : \sigma_1 \sim u_2 : \sigma_2 \mid \phi \\ \Gamma \mid \Psi \vdash v_1 : \mathbb{N} \to \sigma_1 \sim v_2 : \mathbb{N} \to \sigma_2 \mid \forall x_1 x_2 . t_1 = S x_1 \Rightarrow t_2 = S x_2 \Rightarrow \phi[\mathbf{r}_1\ x_1/\mathbf{r}_1][\mathbf{r}_2\ x_2/\mathbf{r}_2]\end{array}}{\Gamma \mid \Psi \vdash \text{case } t_1 \text{ of } 0 \mapsto u_1; S \mapsto v_1 : \sigma_1 \sim \text{case } t_2 \text{ of } 0 \mapsto u_2; S \mapsto v_2 : \sigma_2 \mid \phi}\ \mathsf{NATCASE}$$

$$\frac{\begin{array}{c}\Gamma \mid \Psi \vdash t_1 : \mathsf{list}_{\tau_1} \sim t_2 : \mathsf{list}_{\tau_2} \mid \mathbf{r}_1 = [] \Leftrightarrow \mathbf{r}_2 = [] \\ \Gamma \mid \Psi, t_1 = [], t_2 = [] \vdash u_1 : \sigma_1 \sim u_2 : \sigma_2 \mid \phi \\ \Gamma \mid \Psi \vdash v_1 : \tau_1 \to \mathsf{list}_{\tau_1} \to \sigma_1 \sim v_2 : \tau_2 \to \mathsf{list}_{\tau_2} \to \sigma_2 \mid \\ \forall h_1 h_2 l_1 l_2 . t_1 = h_1 :: l_1 \Rightarrow t_2 = h_2 :: l_2 \Rightarrow \phi[\mathbf{r}_1\ h_1\ l_1/\mathbf{r}_1][\mathbf{r}_2\ h_2\ l_2/\mathbf{r}_2]\end{array}}{\Gamma \mid \Psi \vdash \text{case } t_1 \text{ of } [] \mapsto u_1; \_ :: \_ \mapsto v_1 : \sigma_1 \sim \text{case } t_2 \text{ of } [] \mapsto u_2; \_ :: \_ \mapsto v_2 : \sigma_2 \mid \phi}\ \mathsf{LISTCASE}$$

Fig. E 4. Synchronous case rules

$$\frac{\begin{array}{c}\Gamma \vdash t_1 : \mathbb{B} \\ \Gamma \mid \Psi, t_1 = \mathsf{tt} \vdash u_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi \\ \Gamma \mid \Psi, t_1 = \mathsf{ff} \vdash v_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi\end{array}}{\Gamma \mid \Psi \vdash \text{case } t_1 \text{ of } \mathsf{tt} \mapsto u_1; \mathsf{ff} \mapsto v_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi}\ \mathsf{BOOLCASE-L}$$

$$\frac{\begin{array}{c}\Gamma \vdash t_1 : \mathbb{N} \\ \Gamma \mid \Psi, t_1 = 0 \vdash u_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi \\ \Gamma \mid \Psi \vdash v_1 : \mathbb{N} \to \sigma_1 \sim t_2 : \sigma_2 \mid \forall x_1 . t_1 = S x_1 \Rightarrow \phi[\mathbf{r}_1\ x_1/\mathbf{r}_1]\end{array}}{\Gamma \mid \Psi \vdash \text{case } t_1 \text{ of } 0 \mapsto u_1; S \mapsto v_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi}\ \mathsf{NATCASE-L}$$

$$\frac{\begin{array}{c}\Gamma \vdash t_1 : \mathsf{list}_\tau \\ \Gamma \mid \Psi, t_1 = [] \vdash u_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi \\ \Gamma \mid \Psi \vdash v_1 : \tau \to \mathsf{list}_\tau \to \sigma_1 \sim t_2 : \sigma_2 \mid \forall h_1 l_1 . t_1 = h_1 :: l_1 \Rightarrow \phi[\mathbf{r}_1\ h_1\ l_1/\mathbf{r}_1]\end{array}}{\Gamma \mid \Psi \vdash \text{case } t_1 \text{ of } [] \mapsto u_1; \_ :: \_ \mapsto v_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi}\ \mathsf{LISTCASE-L}$$

Fig. E 5. One-sided case rules

$$\frac{\begin{array}{c}\Gamma \mid \Psi \vdash t_1 : \mathbb{B} \sim t_2 : \mathbb{B} \mid \top \\ \Gamma \mid \Psi, t_1 = \mathsf{tt}, t_2 = \mathsf{tt} \vdash u_1 : \sigma_1 \sim u_2 : \sigma_2 \mid \phi \\ \Gamma \mid \Psi, t_1 \neq \mathsf{tt}, t_2 = \mathsf{tt} \vdash v_1 : \sigma_1 \sim u_2 : \sigma_2 \mid \phi \\ \Gamma \mid \Psi, t_1 = \mathsf{tt}, t_2 \neq \mathsf{tt} \vdash u_1 : \sigma_1 \sim v_2 : \sigma_2 \mid \phi \\ \Gamma \mid \Psi, t_1 \neq \mathsf{tt}, t_2 \neq \mathsf{tt} \vdash v_1 : \sigma_1 \sim v_2 : \sigma_2 \mid \phi\end{array}}{\Gamma \mid \Psi \vdash \mathrm{case}\ t_1\ \mathsf{of}\ \mathsf{tt} \mapsto u_1; \mathsf{ff} \mapsto v_1 : \sigma_1 \sim \mathrm{case}\ t_2\ \mathsf{of}\ \mathsf{tt} \mapsto u_2; \mathsf{ff} \mapsto v_2 : \sigma_2 \mid \phi}\ \mathsf{BBCASE-A}$$

$$\frac{\begin{array}{c}\Gamma \mid \Psi \vdash t_1 : \mathbb{B} \sim t_2 : \mathbb{N} \mid \top \\ \Gamma \mid \Psi, t_1 = \mathsf{tt}, t_2 = 0 \vdash u_1 : \sigma_1 \sim u_2 : \sigma_2 \mid \phi \\ \Gamma \mid \Psi, t_1 \neq \mathsf{tt}, t_2 = 0 \vdash v_1 : \sigma_1 \sim u_2 : \sigma_2 \mid \phi \\ \Gamma \mid \Psi, t_1 = \mathsf{tt} \vdash u_1 : \sigma_1 \sim v_2 : \mathbb{N} \to \sigma_2 \mid \forall x_2. t_2 = S x_2 \Rightarrow \phi[\mathbf{r}_2\ x_2/\mathbf{r}_2] \\ \Gamma \mid \Psi, t_1 \neq \mathsf{tt} \vdash v_1 : \sigma_1 \sim v_2 : \mathbb{N} \to \sigma_2 \mid \forall x_2. t_2 = S x_2 \Rightarrow \phi[\mathbf{r}_2\ x_2/_2]\end{array}}{\Gamma \mid \Psi \vdash \mathrm{case}\ t_1\ \mathsf{of}\ \mathsf{tt} \mapsto u_1; \mathsf{ff} \mapsto v_1 : \sigma_1 \sim \mathrm{case}\ t_2\ \mathsf{of}\ 0 \mapsto u_2; S \mapsto v_2 : \sigma_2 \mid \phi}\ \mathsf{BNCASE-A}$$

$$\frac{\begin{array}{c}\Gamma \mid \Psi \vdash t_1 : \mathbb{B} \sim t_2 : \mathsf{list}_{\tau_2} \mid \top \\ \Gamma \mid \Psi, t_1 = \mathsf{tt}, t_2 = [] \vdash u_1 : \sigma_1 \sim u_2 : \sigma_2 \mid \phi \\ \Gamma \mid \Psi, t_1 \neq \mathsf{tt}, t_2 = [] \vdash v_1 : \sigma_1 \sim u_2 : \tau_2 \to \mathsf{list}_{\tau_2} \to \sigma_2 \mid \phi \\ \Gamma \mid \Psi, t_1 = \mathsf{tt} \vdash u_1 : \sigma_1 \sim v_2 : \tau_2 \to \mathsf{list}_{\tau_2} \to \sigma_2 \mid \forall h_2 l_2. t_2 = h_2 :: l_2 \Rightarrow \phi[\mathbf{r}_2\ h_2\ l_2/\mathbf{r}_2] \\ \Gamma \mid \Psi, t_1 \neq \mathsf{tt} \vdash v_1 : \sigma_1 \sim v_2 : \tau_2 \to \mathsf{list}_{\tau_2} \to \sigma_2 \mid \forall h_2 l_2. t_2 = h_2 :: l_2 \Rightarrow \phi[\mathbf{r}_2\ h_2\ l_2/\mathbf{r}_2]\end{array}}{\Gamma \mid \Psi \vdash \mathrm{case}\ t_1\ \mathsf{of}\ \mathsf{tt} \mapsto u_1; \mathsf{ff} \mapsto v_1 : \sigma_1 \sim \mathrm{case}\ t_2\ \mathsf{of}\ [] \mapsto u_2; \_ :: \_ \mapsto v_2 : \sigma_2 \mid \phi}\ \mathsf{BLCASE-A}$$

$$\frac{\begin{array}{c}\Gamma \mid \Psi \vdash t_1 : \mathbb{N} \sim t_2 : \mathbb{N} \mid \top \\ \Gamma \mid \Psi, t_1 = 0, t_2 = 0 \vdash u_1 : \sigma_1 \sim u_2 : \sigma_2 \mid \phi \\ \Gamma \mid \Psi, t_2 = 0 \vdash v_1 : \mathbb{N} \to \sigma_1 \sim u_2 : \sigma_2 \mid \forall x_1. t_1 = S x_1 \Rightarrow \phi[\mathbf{r}_1\ x_1/\mathbf{r}_1] \\ \Gamma \mid \Psi, t_1 = 0 \vdash u_1 : \sigma_1 \sim v_2 : \mathbb{N} \to \sigma_2 \mid \forall x_2. t_2 = S x_2 \Rightarrow \phi[\mathbf{r}_2\ x_2/\mathbf{r}_2] \\ \Gamma \mid \Psi \vdash v_1 : \mathbb{N} \to \sigma_1 \sim v_2 : \mathbb{N} \to \sigma_2 \mid \forall x_1 x_2. t_1 = S x_1 \Rightarrow t_2 = S x_2 \Rightarrow \phi[\mathbf{r}_1\ x_1/\mathbf{r}_1][\mathbf{r}_2\ x_2/\mathbf{r}_2]\end{array}}{\Gamma \mid \Psi \vdash \mathrm{case}\ t_1\ \mathsf{of}\ 0 \mapsto u_1; S \mapsto v_1 : \sigma_1 \sim \mathrm{case}\ t_2\ \mathsf{of}\ 0 \mapsto u_2; S \mapsto v_2 : \sigma_2 \mid \phi}\ \mathsf{NNCASE-A}$$

$$\frac{\begin{array}{c}\Gamma \mid \Psi \vdash t_1 : \mathsf{list}_{\tau_1} \sim t_2 : \mathsf{list}_{\tau_2} \mid \top \\ \Gamma \mid \Psi, t_1 = [], t_2 = [] \vdash u_1 : \sigma_1 \sim u_2 : \sigma_2 \mid \phi \\ \Gamma \mid \Psi, t_2 = [] \vdash v_1 : \tau_1 \to \mathsf{list}_{\tau_1} \to \sigma_1 \sim u_2 : \sigma_2 \mid \forall h_1 l_1. t_1 = h_1 :: l_1 \Rightarrow \phi[\mathbf{r}_1\ h_1\ l_1/\mathbf{r}_1] \\ \Gamma \mid \Psi, t_1 = [] \vdash u_1 : \tau_1 \to \mathsf{list}_{\tau_1} \to \sigma_1 \sim v_2 : \tau_2 \to \mathsf{list}_{\tau_2} \to \sigma_2 \mid \\ \forall h_2. t_2 = h_2 :: l_2 \Rightarrow \phi[\mathbf{r}_2\ h_2\ l_2/\mathbf{r}_2] \\ \Gamma \mid \Psi \vdash v_1 : \tau_1 \to \mathsf{list}_{\tau_1} \to \sigma_1 \sim v_2 : \tau_2 \to \mathsf{list}_{\tau_2} \to \sigma_2 \mid \\ \forall h_1 h_2 l_1 l_2. t_1 = h_1 :: l_1 \Rightarrow t_2 = h_2 :: l_2 \Rightarrow \phi[\mathbf{r}_1\ h_1\ l_1/\mathbf{r}_1][\mathbf{r}_2\ h_2\ l_2/\mathbf{r}_2]\end{array}}{\Gamma \mid \Psi \vdash \mathrm{case}\ t_1\ \mathsf{of}\ [] \mapsto u_1; \_ :: \_ \mapsto v_1 : \sigma_1 \sim \mathrm{case}\ t_2\ \mathsf{of}\ [] \mapsto u_2; \_ :: \_ \mapsto v_2 : \sigma_2 \mid \phi}\ \mathsf{LLCASE-A}$$

Fig. E 6. Asynchronous case rules (selected)

$$\frac{\begin{array}{c}\Gamma \mid \Psi \vdash t_1 : \mathbb{N} \sim t_2 : \mathbb{N} \mid \phi' \wedge \mathbf{r}_1 = 0 \Leftrightarrow \mathbf{r}_2 = 0 \\ \Gamma \mid \Psi, \phi'[0/\mathbf{r}_1][0/\mathbf{r}_2] \vdash u_1 : \sigma_1 \sim u_2 : \sigma_2 \mid \phi \\ \Gamma \mid \Psi \vdash v_1 : \mathbb{N} \to \sigma_1 \sim v_2 : \mathbb{N} \to \sigma_2 \mid \forall x_1 x_2 . \phi'[S x_1/\mathbf{r}_1][S x_2/\mathbf{r}_2] \Rightarrow \phi[\mathbf{r}_1\ x_1/\mathbf{r}_1][\mathbf{r}_2\ x_2/\mathbf{r}_2]\end{array}}{\Gamma \mid \Psi \vdash \text{case } t_1 \textsf{ of } 0 \mapsto u_1 ; S \mapsto v_1 : \sigma_1 \sim \text{case } t_2 \textsf{ of } 0 \mapsto u_2 ; S \mapsto v_2 : \sigma_2 \mid \phi} \textsf{NATCASE}*$$

$$\frac{\begin{array}{c}\Gamma \mid \Psi \vdash t_1 : \mathsf{list}_{\tau_1} \sim t_2 : \mathsf{list}_{\tau_2} \mid \phi' \wedge \mathbf{r}_1 = [] \Leftrightarrow \mathbf{r}_2 = [] \\ \Gamma \mid \Psi, \phi'[[]/\mathbf{r}_1][[]/\mathbf{r}_2] \vdash u_1 : \sigma_1 \sim u_2 : \sigma_2 \mid \phi \\ \Gamma \mid \Psi \vdash v_1 : \tau_1 \to \mathsf{list}_{\tau_1} \to \sigma_1 \sim v_2 : \tau_2 \to \mathsf{list}_{\tau_2} \to \sigma_2 \mid \\ \forall h_1 h_2 l_1 l_2 . \phi'[h_1 :: l_1/\mathbf{r}_1][h_2 :: l_2/\mathbf{r}_2] \Rightarrow \phi[\mathbf{r}_1\ h_1\ l_1/\mathbf{r}_1][\mathbf{r}_2\ h_2\ l_2/\mathbf{r}_2]\end{array}}{\Gamma \mid \Psi \vdash \text{case } t_1 \textsf{ of } [] \mapsto u_1 ; \_ :: \_ \mapsto v_1 : \sigma_1 \sim \text{case } t_2 \textsf{ of } [] \mapsto u_2 ; \_ :: \_ \mapsto v_2 : \sigma_2 \mid \phi} \textsf{LISTCASE}*$$

Fig. E 7. Alternative case rules

$$\frac{\begin{array}{c}\mathcal{D}ef(f_1, x_1, e_1)\ \mathcal{D}ef(f_2, x_2, e_2)\ x_1, f_1 \notin FV(e_2)\ x_2, f_2 \notin FV(e_1) \\ \Gamma, x_1 : I_1, x_2 : I_2, f_1 : I_1 \to \sigma, f_2 : I_2 \to \sigma_2 \mid \Psi, \phi', \\ \forall m_1 m_2 . (|m_1|, |m_2|) < (|x_1|, |x_2|) \Rightarrow \phi'[m_1/x_1][m_2/x_2] \Rightarrow \phi[m_1/x_1][m_2/x_2][f_1\ m_1/\mathbf{r}_1][f_2\ m_2/\mathbf{r}_2] \vdash \\ e_1 : \sigma_1 \sim e_2 : \sigma_2 \mid \phi\end{array}}{\Gamma \mid \Psi \vdash \text{letrec } f_1\ x_1\ = e_1 : I_1 \to \sigma_2 \sim \text{letrec } f_2\ x_2\ = e_2 : I_2 \to \sigma_2 \mid \forall x_1 x_2 . \phi' \Rightarrow \phi[\mathbf{r}_1\ x_1/\mathbf{r}_1][\mathbf{r}_2\ x_2/\mathbf{r}_2]} \textsf{LETREC}$$

$$\frac{\begin{array}{c}\mathcal{D}ef(f_1, x_1, e_1)\ x_1, f_1 \notin FV(e_2) \\ \Gamma, x_1 : I_1, f_1 : I_1 \to \sigma \mid \Psi, \phi', \\ \forall m_1 . |m_1| < |x_1| \Rightarrow \phi'[m_1/x_1] \Rightarrow \phi[m_1/x_1][m_2/x_2][f_1\ m_1/\mathbf{r}_1][t_2/\mathbf{r}_2] \vdash e_1 : \sigma_1 \sim t_2 : \sigma_2 \mid \phi\end{array}}{\Gamma \mid \Psi \vdash \text{letrec } f_1\ x_1\ = e_1 : I_1 \to \sigma_2 \sim t_2 : \sigma_2 \mid \forall x_1 . \phi' \Rightarrow \phi[\mathbf{r}_1\ x_1/\mathbf{r}_1]} \textsf{LETREC}-\textsf{L}$$

$$\text{where } I_1, I_2 \in \{\mathbb{N}, \mathsf{list}_\tau\}$$

Fig. E 8. Recursion rules