Implicit Incremental Computation for CostIt

Internship Report

Zoe Paraskevopoulou, supervised by Deepak Garg, Max Planck Institute of Software Systems

August 21, 2015

The general context

Several applications manipulate data that change dynamically over time, and only a small change in the data may trigger the re-computation of the whole output. However, it is often the case that only a small part of the output is affected by a change in the application data. That raises an interesting question: could we reuse parts of previous computations in order to increase the performance of software applications? The aim of incremental computation is to reuse the results of earlier runs in order to recompute only those outputs that depend on data that has changed, avoiding recomputation of data that is not affected by the changes [1]. We refer to the process of updating the parts of the output that depends on data that have changed as *change propagation*. Incremental computation may be explicit or implicit. In the former the programmer should explicitly write code that performs change propagation, while in the latter no manual effort is required from the programmer. The original program can be compiled into a *self-adjusting* program, i.e. a program that can propagate changes from the inputs to the output, without recomputing the parts of the result that remain the same.

The research problem

Various techniques have been proposed in order to derive software that responds automatically and efficiently to changing data, including static and static and dynamic dependency graphs[6, 18, 2] and function call memoization[15, 10], in which change propagation may perform asymptotically faster than recomputing the output from scratch. However, in all previous work on incremental computation, the programmer must reason manually about the complexity of an incremental run, by analysis of the cost semantics of program. CostIt is a higher-order functional language equipped with a refinement type system that allows the programmer to prove upper bounds for the cost of change propagation. That is made possible using index refinement types, in the style of DML [17], and type annotations that allows tracking which part of the data may or may not change after an update.

Contribution

This internship focuses on proving the soundness of the type system with respect to a concrete semantics of change propagation. More specifically, we prove that the cost incurred by the type system is a sound approximation, i.e. an upper bound, on the cost of propagating changes in a program after a change in its inputs. We propose a translation scheme that given a source program written in CostIt [5] transforms it to a self-adjusting program written in a target language that provides support for change propagation. To this end, we enrich an ML-like language with new primitives and a modified runtime in order to facilitate change propagation. Our target language, referred to as saML, allows us to propagate changes in programs by reusing parts of the data computed in the first run and only recomputing and updating in place data that may have changed. During the first run we record the components that may have changed along with a closure that computes their updated value. In each incremental run it suffices to only execute the recorded closures and update the changeable components.

Arguments supporting the validity of our approach

We prove that our translation is type preserving, i.e. that the translated program is well-typed in the target language and that its type coincides with the translation of the source type into a target type. Furthermore, we prove the correctness of our translation by proving contextual equivalence between a from-scratch execution in the source and a from-scratch execution in the target language. Similarly, we prove that our change propagation mechanism is sound by proving contextual equivalence between an incremental run in the target language and a from-scratch execution of the same program in the source language. More importantly, we prove that the cost incurred by the type system soundly approximates the cost of an incremental run of the self-adjusting saML program. That establishes the soundness of the type system. [1]. In order to show the correctness of the translation we construct a step-indexed Kripke relational model, a powerful proof technique with well-studied applications in proving program equivalence and correctness of program transformations [3, 7, 16].

Summary and future work

The main contribution of this internship is the soundness proof of CostIt, a novel proof system that allows to prove upper bounds on the complexity of change propagation. This is done by translating a CostIt program to a self-adjusting ML program and proving that the cost incurred by the type system is a sound approximation of the actual change propagation cost.

There is ongoing work, by Ezgi Çiçek and Deepak Garg, towards the implementation of the type system using bidirectional type checking, a combination of type inference and type checking that reduces the amount of type annotations that the programmer has to write. The domain of constraints is intractable and thus we cannot expect fully automated verification. With this in mind, it is worth exploring the possibility of building a semi-automatic interactive prover using a combination of automatic and manual techniques, such as constraint solvers and interactive provers, as in Why3.

Another possible future direction, inspired by recent work by Hammer *et al.* [9], would be to modify CostIt's static and dynamic semantics in order to model demand-driven incremental computation. Unlike traditional incremental computation, demand-driven incremental computation defers the re-computation of a result until the result is used, allowing for even better speedups over full re-evaluation. Similar to traditional change propagation, no formal system that supports reasoning about demand-driven change propagation cost has been proposed up to now.

Acknowledgments

I wish to thank my advisor, Deepak Garg, for giving me the opportunity to join MPI as an intern, for his continuous guidance, assistance and encouragement and also his vast knowledge. I would like to thank my collaborator Ezgi Çiçek for the seamless and fruitful collaboration and her willingness to share insights. I would also like to thank the students and the faculty of the PL/V group for their eagerness to provide help and feedback. Last but not least, I would like to thank Nick Giannarakis for his endless support and insightful discussions.

This internship was funded by an INRIA/MPRI scholarship.

[1] The appendix with our proofs is available here

1 Introduction

CostIt is a functional language equipped with a type system that is expressive enough to facilitate reasoning about the *dynamic stability* of programs [5], i.e. the time requires to propagate changes from the inputs to the output of a program. This is made possible by treating dynamic stability as an effect in the type system [13]. Furthermore, the cost of change propagation can depend on attributes of the input, such as the length of a list. In order to track such attributes of program inputs the type system is equipped with index refinement types, in the style of DML [17] or DFuzz [8].

Unlike the previous version of CostIt [5], that disallowed control flow changes between the initial and a subsequent run by enforcing that control flow does not depend on data that may change, in this work we consider an extension of CostIt that allows for control flow changes between the initial and a subsequent run. In order to facilitate control flow dependency from changeable data the type system is extended with a mode that allows to establish upper bounds for worst case from-scratch execution time. Reasoning about from-scratch execution is required as the program may follow paths that were not explored during the first run and thus there is not sufficient runtime information to perform change propagation. This extension is unpublished work by Ezgi Çiçek and Deepak Garg.

CostIt's typing judgments have the form $\vdash_{\epsilon}^{\kappa} e : \tau$, where ϵ denotes the typing mode and κ the upper bound of the cost of the computation. When the typing is in change propagation mode, i.e. ϵ is $\mathbb{S}$, then the derived cost represents the upper bound on cost of change propagation. Correspondingly, when the typing is in from-scratch mode, i.e. ϵ is $\mathbb{C}$, the derived mode is the worst case execution time. Similarly, functions are annotated with effects, which represent either dynamic stability or worst case execution time. A function of type $\tau_1 \xrightarrow{\mathbb{S}(\kappa)} \tau_2$ can propagate changes with cost less or equal to κ. A function of type $\tau_1 \xrightarrow{\mathbb{C}(\kappa)} \tau_2$ can execute from-scratch with cost less or equal to κ.

In order to differentiate between changeable and stable values, types are annotated with labels that indicate whether a value can change or not. Similar annotations have also been used in the context of information flow control analysis [14], binding-time analysis [13], but also incremental computation [4]. A value that can change between two runs has a type annotated with $\mathbb{C}\cdot$. A value that cannot change between different runs is annotated with $(\cdot)^{\mathbb{S}}$ or $(\cdot)^{\square}$. Intuitively, $(\cdot)^{\mathbb{S}}$ and $(\cdot)^{\square}$ differ only in higher-order and sum types. The definition of a function that is labeled with $(\cdot)^{\mathbb{S}}$ should remain the same between two runs. Nevertheless, the definition may capture variables that can change. A function labeled with $(\cdot)^{\square}$ not only itself cannot change but also does not capture inside its body any changeable variable from the environment. Similarly, a sum type annotated with $(\cdot)^{\square}$ should not capture inside its definition values that are allowed to change between different runs.

In the following subsections we will describe the type system first through an example and then through the formal presentation of the typing rules.

1.1 Typing by Example

As our running example we will pick the higher-order function `map`. For simplicity, we will assume that it operates on lists of integers. In CostIt `map` can be given the type

$$\forall \kappa \overset{\mathbb{S}(0)}{::} \dot{\mathbb{R}}^{+}.\ ((\texttt{int})^{\mathbb{C}} \xrightarrow{\mathbb{S}(\kappa)} (\texttt{int})^{\mathbb{C}})^{\square} \xrightarrow{\mathbb{S}(0)} \forall n \overset{\mathbb{S}(0)}{::} \dot{\mathbb{N}}.\ \forall a \overset{\mathbb{S}(0)}{::} \dot{\mathbb{N}}.\ \texttt{list}\,[n]^{a}\ (\texttt{int})^{\mathbb{C}} \xrightarrow{\mathbb{S}(\kappa\cdot\alpha)} \texttt{list}\,[n]^{\alpha}\ (\texttt{int})^{\mathbb{C}}$$

The code of the function can be seen in fig. 6. The type reads follows: given a function, that itself does not change, does not capture any changeable variables from the environment, and executes from scratch with cost κ, and a list with n elements from which at most α can change, the function can propagate changes in the input list to the output list with cost at most $\kappa \cdot \alpha$.

We will explain how this type can be derived in the type system. When the list is empty the computation has zero cost and the result is derived trivially. Assume now that the list has length $n + 1$. There are two cases: a.) the head is changeable and b.) the head is stable. In the first case, the application of the function to the head of the lists will incur cost κ and, furthermore, the rest of the list has type $\texttt{list}\,[n-1]^{\alpha-1}\ \tau$. By the induction hypothesis we can derive that the recursive call incurs cost $\kappa * (\alpha - 1)$ and the result

$$\begin{array}{llll}
\text{Base Types} & A & ::= & \texttt{int} \mid \tau_1 + \tau_2 \mid \tau_1 \xrightarrow{\mathbb{S}(\kappa)} \tau_2 \mid \tau_1 \xrightarrow{\mathbb{C}(\kappa)} \tau_2 \mid \forall i \overset{\mathbb{S}(\kappa)}{::} S.\, \tau \mid \forall i \overset{\mathbb{C}(\kappa)}{::} S.\, \tau \\
\text{Types} & \tau & ::= & (A)^{\mu} \mid \texttt{unit} \mid \tau_1 \times \tau_2 \mid \texttt{list}\,[n]^{\alpha}\; \tau \mid \exists i.\, \tau \mid C \to \tau \mid C \wedge \tau \mid \\
\text{Modes} & \mu, \epsilon, \delta & ::= & \mathbb{S} \mid \mathbb{C} \mid \square \\
\text{Sorts} & S & ::= & \dot{\mathbb{N}} \mid \dot{\mathbb{R}}^{+} \\
\text{Index terms} & I, \kappa & ::= & i \mid 0 \mid I \mathbin{\hat{+}} 1 \\
 & & & I_1 \mathbin{\hat{+}} I_2 \mid I_1 \mathbin{\hat{-}} I_2 \mid \hat{\frac{I_1}{I_2}} \mid I_1 \mathbin{\hat{\cdot}} I_2 \mid \lceil \hat{I} \rceil \mid \lfloor \hat{I} \rfloor \mid \mathtt{l\hat{o}g}_2(I) \mid I_1^{I_2} \mid \\
 & & & \mathtt{m\hat{i}n}(I_1, I_2) \mid \mathtt{m\hat{a}x}(I_1, I_2) \mid \hat{\sum}_{i=I_1}^{I_n} I \mid C \mathbin{?} I_1 \;:\; I_2 \\
\text{Constraints} & C & ::= & I_1 \doteq I_2 \mid I_1 \mathbin{\dot{<}} I_2 \mid \dot{\neg} C \mid \dot{\bot} \mid C_1 \mathbin{\dot{\wedge}} C_2 \mid C_1 \mathbin{\dot{\vee}} C_2 \\
\text{Constraint env.} & \Phi & ::= & \dot{\top} \mid \Phi \mathbin{\dot{\wedge}} C \\
\text{Sort env.} & \Delta & ::= & \emptyset \mid \Delta, i :: S \\
\text{Type env.} & \Gamma & ::= & \emptyset \mid \Gamma, x : \tau
\end{array}$$

Figure 1: Types

follows immediately. The second case is less trivial and requires non-standard reasoning. Since the head is stable the rest of the list will have type $\texttt{list}\,[n-1]^{\alpha}$ and from the induction hypothesis we derive that the recursive call incurs cost $\kappa * \alpha$. It suffices to show that the function application will incur zero cost and the result of the application, that is cons-ed to the list, is stable. Since there are no free variables that can change between runs in the expression or variables that can capture changeable values, it should be the case that the value of the expression cannot change between runs and thus, no change propagation is needed. Furthermore, the result can be safely annotated as stable, and thus added to the list without increasing the count of changeable elements. This form of reasoning is embedded as a typing rule in the type system, and as we will explain later in this section, corresponds to a comonadic reasoning principle.

1.2 Syntax and Semantics

In this subsection we will describe the syntax and the semantics of CostIt's language. The complete syntax of types and terms can be seen in figs. 1 and 2.

$$\begin{array}{llll}
\text{Values} & v & ::= & \mathtt{n} \mid \mathtt{b} \mid (v_1, v_2) \mid \texttt{inl}\; v \mid \texttt{inr}\; v \mid [] \mid v_1 :: v_2 \mid \\
 & & & \texttt{fix}\; f(x).\, e \mid \Lambda.\, e \mid \texttt{pack}\; v \mid () \\
\text{Expressions} & e, f & ::= & x \mid \mathtt{n} \mid \mathtt{b} \mid (e_1, e_2) \mid \texttt{fst}\; e \mid \texttt{snd}\; e \mid \texttt{inl}\; e \mid \texttt{inr}\; e \mid \texttt{case}(e, x.y, e_1.e_2) \mid \\
 & & & [] \mid e_1 :: e_2 \mid (\texttt{case}_{\mathtt{L}}\; e\; \texttt{of nil} \;\to\; e_1 \mid \texttt{cons}(h, tl) \;\to\; e_2) \mid \\
 & & & \texttt{fix}\; f(x).\, e \mid e_1\; e_2 \mid \zeta(e) \mid \Lambda.\, e \mid e[] \mid \texttt{pack}\; e \mid \texttt{unpack}\; e\; \texttt{as}\; x\; \texttt{in}\; e' \mid \\
 & & & \texttt{let}\; x \;=\; e_1\; \texttt{in}\; e_2 \mid ()
\end{array}$$

Figure 2: Value and expression syntax

Index terms can be either natural numbers or positive real numbers, represented with the sorts $\dot{\mathbb{N}}$ and $\dot{\mathbb{R}}^{+}$ respectively. They are equipped with the standard arithmetical operations that are overloaded for the two sorts and there is an implicit coercion from $\dot{\mathbb{N}}$ to $\dot{\mathbb{R}}^{+}$. The sorting judgment $\Delta \vdash I :: S$, where Δ is the index environment assigning sorts to free index variables, assigns a sort to an index term.

Constraints are propositions over index terms. They are subject to a standard well-formedness judgment, denoted $\Delta \vdash C\; \texttt{wf}$. Moreover, we define a logical entailment judgment, denoted $\Delta; \Phi \models C$ where Φ is the environment where we collect constraints, which is defined by interpreting constraints in arithmetic. The sorting and well-formedness judgments for index terms and constraints are similar to those presented by Çiçek *et al.* [5] thus they are omitted.

The types of CostIt are simple types refined by index terms and annotated with changeability annotations. Functions are annotated with effects as described earlier in this section. Universally quantified types are also annotated with effects that represent the cost of change propagation or from-scratch execution of the closure. Lists are refined with both the length of the list and the number of elements that are allowed to change between runs. More precisely the list type $\texttt{list}\,[n]^{\alpha}\;\tau$ denotes a list of type τ with n elements from which at α can change.

The syntax of CostIt's terms is fairly standard. Note that abstraction and instantiation of index terms, denoted $\Lambda.\,e$ and $e[]$ respectively, do not mention index terms to avoid syntactic. For more details the reader can refer to [5].

Similar to all other refinement type systems, CostIt is equipped with a subtyping judgment, denoted $\Delta;\Phi \vdash_{\epsilon}^{\kappa} \tau_1 \sqsubseteq \tau 2$. Subtyping has two subtyping modes, one for from-scratch evaluation and the other for change propagation. The only difference in the two modes is the cost. We will not explain the intuition behind the derived cost in this section. We will defer this discussion until section 4. Selected subtyping rules are presented in fig. 3.

$$\frac{\Delta;\Phi \models \kappa \;\hat{+}\; \texttt{cost}_{\texttt{conv}}(\tau_1^{\downarrow\Box},\tau_1) \;\hat{+}\; (\delta = \mathbb{C}) \;?\; (1 \;\hat{+}\; \texttt{cost}_{\texttt{conv}}(\tau_2,\tau_2^{\downarrow\Box})) \;:\; 0 \;\dot{\leq}\; \kappa'}{\Delta;\Phi \vdash_{\epsilon}^{0} (\tau_1 \xrightarrow{\delta(\kappa)} \tau_2)^{\Box} \sqsubseteq (\tau_1^{\downarrow\Box} \xrightarrow{\delta(\kappa')} \tau_2^{\downarrow\Box})^{\Box}} \to 1$$

$$\frac{\Delta;\Phi \vdash_{\delta}^{\kappa_1} \tau_1' \sqsubseteq \tau_1 \qquad \Delta;\Phi \vdash_{\delta}^{\kappa_2} \tau_2 \sqsubseteq \tau_2' \qquad \Delta;\Phi \models \kappa \;\hat{+}\; \kappa_1 \;\hat{+}\; \kappa_2 \;\hat{+}\; (\delta = \mathbb{C}) \;?\; 1 \;:\; 0 \leq \kappa'}{\Delta;\Phi_A \vdash_{\epsilon}^{0} \tau_1 \xrightarrow{\delta(\kappa)} \tau_2 \sqsubseteq \tau_1' \xrightarrow{\delta(\kappa')} \tau_2'} \to 2$$

$$\frac{\Delta;\Phi \models \kappa \doteq (\epsilon = \mathbb{C}) \;?\; 1 \;\hat{+}\; \hat{\texttt{max}}(\texttt{cost}_{\texttt{conv}}(\tau_1,\tau_1^{\downarrow\Box}), \texttt{cost}_{\texttt{conv}}(\tau_2,\tau_2^{\downarrow\Box})) \;:\; 0}{\Delta;\Phi \vdash_{\epsilon}^{\kappa} (\tau_1 + \tau_2)^{\Box} \sqsubseteq (\tau_1^{\downarrow\Box} + \tau_2^{\downarrow\Box})^{\Box}} \;\mathbf{+1}$$

$$\frac{\Delta;\Phi \models \alpha \;\doteq\; 0}{\Delta;\Phi \vdash_{\epsilon}^{0} \texttt{list}\,[n]^{\alpha}\;\tau \sqsubseteq \texttt{list}\,[n]^{\alpha}\;\tau^{\downarrow\Box}} \;\mathbf{l1*}$$

$$\frac{\Delta;\Phi \models n \doteq n' \qquad \Delta;\Phi \models \alpha \doteq \alpha' \qquad \Delta;\Phi \vdash_{\mathbb{S}}^{\kappa'} \tau \sqsubseteq \tau' \qquad \Delta;\Phi \models \kappa \doteq \alpha \;\hat{\cdot}\; \kappa'}{\Delta;\Phi \vdash_{\mathbb{S}}^{\kappa} \texttt{list}\,[n]^{\alpha}\;\tau \sqsubseteq \texttt{list}\,[n']^{\alpha'}\;\tau'} \;\mathbf{l2\text{-}}\mathbb{S}$$

$$\frac{\Delta;\Phi \models \mu \leq \mu' \qquad \Delta;\Phi \models \kappa \doteq \texttt{cost}_{\texttt{conv}}((A)^{\mu},(A)^{\mu'})}{\Delta;\Phi \vdash_{\epsilon}^{\kappa} (A)^{\mu} \sqsubseteq (A)^{\mu'}} \;\mu \qquad \frac{\Delta;\Phi_A \vdash_{\epsilon}^{\kappa} A \sqsubseteq A' \qquad \Delta;\Phi \models \kappa \doteq \kappa' \;\hat{+}\; (\mu = \mathbb{C}) \;?\; 2 \;:\; 0}{\Delta;\Phi \vdash_{\epsilon}^{\kappa} (A)^{\mu} \sqsubseteq (A')^{\mu}} \;\mathbf{C}$$

$$\frac{\Delta;\Phi \vdash_{\epsilon}^{\kappa_1} \tau_1 \sqsubseteq \tau_2 \qquad \Delta;\Phi \vdash_{\epsilon}^{\kappa_2} \tau_2 \sqsubseteq \tau_3 \qquad \Delta;\Phi \models \kappa \doteq \kappa_1 \;\hat{+}\; \kappa_2}{\Delta;\Phi \vdash_{\epsilon}^{\kappa} \tau_1 \sqsubseteq \tau_3} \mathbf{tran}$$

Figure 3: Selected subtyping rules

The rule → 1 states if a function that does not change and does not capture changeable variables from its environment is given an argument that does not change nor capture changeable variables, then the result also cannot change or capture changeable variables. The rule → 2 states that subtyping is contravariant in the argument type and covariant in the result type. Subtyping is also a covariant in the cost of the function, since it represents an upper bound. The rule **l1** states that the type of a list with that contains only stable elements can be stabilized. The type stabilization function, denoted $(\cdot)^{\downarrow\Box}$, is defined as the homomorphic lifting on types of the operation that turns all the changeability annotations to $(\cdot)^{\Box}$, i.e. $(A^{\mu})^{\downarrow\Box} = (A)^{\Box}$. Note that this operation leaves base types unaffected. The rule **l2**$\mathbb{S}$ states that if a type is a subtype of an other type then a list of elements of the first type is a subtype of list of elements of the second type. The rule μ states that a stable type can be lifted to a changeable type. [2]

The typing judgment of CostIt has the form $\Delta;\Phi;\Gamma \vdash_{\epsilon}^{\kappa} e : \tau$, where Δ is the sorting environment, Φ is the constraint environment, Γ is the typing environment, ϵ is the mode and κ the dynamic stability. Selected

[2] Changeability annotations enjoy a total order, namely $\Box < \mathbb{S} < \mathbb{C}$

subtyping rules are presented in fig. 4. Again, we will defer the explanation of the derived dynamic stability until section 4.

The rule **fix1** states that a function is typable with a change propagation (resp. from-scratch) cost if its body is typable in change propagation (resp. from-scratch) mode incurring the same cost. Rule **app** states that a function can be applied in mode ϵ if its mode is at least ϵ, i.e. a change propagation function cannot be applied in from scratch mode. Furthermore, it states that a function that changes between two runs can only re-execute from scratch. The **case** rule is straight forward. An interesting detail is that when the scrutinee can change then the branches should be typed in from-scratch mode, reflecting the fact they should be re-evaluated during change propagation. Rule **nochange** embodies the co-monadic reasoning principle in the type system treating $(\tau)^{\square}$ as a co-monadic type [12]. It states that if an expression is typable in a stable context, i.e. if all of its free variables cannot change between two runs, then change propagation can be bypassed incurring zero cost. This relies on a very intuitive property of change propagation; if none of an expressions's free variables can change between two runs then the expression should evaluate to the same value. Furthermore, the type of the expression can be stabilized, reflecting the fact that it cannot change between runs. The rule **caseL** embeds the inductive reasoning principle we used in the example we gave earlier in the type system. It has tree premises. The first premise applies when the scrutinee evaluates to an empty list. The second and the third cases correspond to the inductive case. The second applies when the head of the list is stable. In this case the total number of elements in the tail is decreased by one. The third premise applies when the head of the list is changeable. Both the total number of elements in the list and the total number of changeable elements in the tail are decreased by one.

$$\dfrac{\Delta;\Phi;\Gamma, f:(\tau_1 \xrightarrow{\delta(\kappa)} \tau_2)^{\mathbb{S}}, x:\tau_1 \vdash^{\kappa}_{\delta} e:\tau_2}{\Delta;\Phi;\Gamma \vdash^{0}_{\epsilon} \texttt{fix}\ f(x).\, e : (\tau_1 \xrightarrow{\delta(\kappa)} \tau_2)^{\mathbb{S}}}\ \textbf{fix1}$$

$$\dfrac{\begin{array}{c}\Delta;\Phi;\Gamma \vdash^{\kappa_1}_{\epsilon} e_1 : (\tau_1 \xrightarrow{\delta(\kappa')} \tau_2)^{\mu} \qquad \Delta;\Phi;\Gamma \vdash^{\kappa_2}_{\epsilon} e_2 : \tau_1 \qquad (\epsilon \sqcup \mu) \leq \delta \qquad \mu \trianglelefteq \tau_2 \\ \Delta;\Phi \models \kappa \doteq \kappa' \mathbin{\hat{+}} \kappa_1 \mathbin{\hat{+}} \kappa_2 \mathbin{\hat{+}} ((\epsilon = \mathbb{C})\ ?\ 1\ :\ 0) \mathbin{\hat{+}} ((\mu = \mathbb{C})\ ?\ 1\ :\ 0) \mathbin{\hat{+}} ((\epsilon = \mathbb{S} \wedge \mu = \mathbb{C})\ ?\ \texttt{cost}_{\texttt{deepref}}(\tau_2) + 2\ :\ 0)\end{array}}{\Delta;\Phi;\Gamma \vdash^{\kappa}_{\epsilon} e_1\ e_2 : \tau_2}\ \textbf{app}$$

$$\dfrac{\begin{array}{c}\Delta;\Phi;\Gamma \vdash^{\kappa_e}_{\epsilon} e : (\tau_1 + \tau_2)^{\mu} \qquad \Delta;\Phi;\Gamma, x:\tau_1 \vdash^{\kappa'}_{\epsilon \sqcup \mu} e_1 : \tau \qquad \Delta;\Phi;\Gamma, y:\tau_2 \vdash^{\kappa'}_{\epsilon \sqcup \mu} e_2 : \tau \\ \Delta;\Phi \models \kappa \doteq \kappa_e \mathbin{\hat{+}} \kappa' \mathbin{\hat{+}} ((\epsilon = \mathbb{C})\ ?\ 1\ :\ 0) \mathbin{\hat{+}} ((\mu = \mathbb{C})\ ?\ 1\ :\ 0) \mathbin{\hat{+}} ((\epsilon = \mathbb{S} \wedge \mu = \mathbb{C})\ ?\ \texttt{cost}_{\texttt{deepref}}(\tau) + 2\ :\ 0)\end{array}}{\Delta;\Phi;\Gamma \vdash^{\kappa}_{\epsilon} \texttt{case}(e, x.e_1, y.e_2) : \tau}\ \textbf{case}$$

$$\dfrac{\Delta;\Phi;\Gamma \vdash^{\kappa_1}_{\epsilon} e : \tau' \qquad \Delta;\Phi \vdash^{\kappa_2}_{\epsilon} \tau' \sqsubseteq \tau \hookrightarrow \underline{\mathbf{g}} \qquad \Delta;\Phi \models \kappa_1 \mathbin{\hat{+}} \kappa_2 \mathrel{\dot{\leq}} \kappa}{\Delta;\Phi;\Gamma \vdash^{\kappa}_{\epsilon} e : \tau}\ \sqsubseteq$$

$$\dfrac{\Delta;\Phi;\Gamma \vdash^{\kappa'}_{\epsilon} e : \tau \qquad \forall x \in \Gamma\ \ \Delta;\Phi \vdash \Gamma(x) \sqsubseteq \Gamma(x)^{\downarrow\square} \qquad \Delta;\Phi \models \kappa \doteq (\epsilon = \mathbb{S}\ ?\ 0\ :\ \kappa')}{\Delta;\Phi;\Gamma,\Gamma' \vdash^{\kappa}_{\epsilon} e : \tau^{\downarrow\square}}\ \textbf{nochange}$$

$$\dfrac{\Delta;\Phi;\Gamma \vdash^{\kappa_1}_{\epsilon} e_1 : \tau^{\downarrow\square} \qquad \Delta;\Phi;\Gamma \vdash^{\kappa_2}_{\epsilon} e_2 : \texttt{list}\,[n]^{\alpha}\ \tau \qquad \Delta;\Phi \models \kappa \doteq \kappa_1 \mathbin{\hat{+}} \kappa_2}{\Delta;\Phi;\Gamma \vdash^{\kappa}_{\epsilon} e_1 :: e_2 : \texttt{list}\left[n \mathbin{\hat{+}} 1\right]^{\alpha}\ \tau}\ \textbf{cons1}$$

$$\dfrac{\Delta;\Phi;\Gamma \vdash^{\kappa_1}_{\epsilon} e_1 : \tau \qquad \Delta;\Phi;\Gamma \vdash^{\kappa_2}_{\epsilon} e_2 : \texttt{list}\,[n]^{\alpha-1}\ \tau \qquad \Delta;\Phi \models \alpha \mathrel{\dot{>}} 0 \qquad \Delta;\Phi \models \kappa \doteq \kappa_1 \mathbin{\hat{+}} \kappa_2}{\Delta;\Phi;\Gamma \vdash^{\kappa}_{\epsilon} e_1 :: e_2 : \texttt{list}\left[n \mathbin{\hat{+}} 1\right]^{\alpha}\ \tau}\ \textbf{cons2}$$

$$\dfrac{\begin{array}{c}\Delta;\Phi;\Gamma \vdash^{\kappa_e}_{\epsilon} e : \texttt{list}\,[n]^{\alpha}\ \tau \\ \Delta;\Phi \wedge n\ \doteq\ 0;\Gamma \vdash^{\kappa'}_{\epsilon} e_1 : \tau' \qquad \Delta, i :: \iota;\Phi \wedge n \doteq i+1;\Gamma, h:\tau^{\downarrow\square}, tl : \texttt{list}\,[i]^{\alpha}\ \tau \vdash^{\kappa'}_{\epsilon} e_2 : \tau' \\ \Delta, i :: \iota, \beta :: \iota;\Phi \wedge n \doteq i+1 \wedge \alpha \doteq \beta + 1;\Gamma, h:\tau, tl : \texttt{list}\,[i]^{\beta}\ \tau \vdash^{\kappa'}_{\epsilon} e_2 : \tau' \qquad \Delta;\Phi \models \kappa \doteq \kappa_e \mathbin{\hat{+}} \kappa' \mathbin{\hat{+}} (\epsilon = \mathbb{C})\ ?\ 1\ :\ 0\end{array}}{\Delta;\Phi;\Gamma \vdash^{\kappa}_{\epsilon} \texttt{case}_{\texttt{L}}\ e\ \texttt{of}\ \texttt{nil}\ \to\ e_1 \mid \texttt{cons}(h, tl)\ \to\ e_2 : \tau'}\ \textbf{caseL}$$

Figure 4: Selected tying rules

The dynamic semantics are the standard call-by-value semantics thus their detailed presentation is omitted. The big step evaluation judgment is modified in order to return the cost of the evaluation. Note

$$\begin{tikzcd}
\vdash^{\kappa}_{\epsilon} e : \tau \arrow[r, "\text{evaluates to}"] \arrow[d, "\text{translates to}"] & v_1 \arrow[d, phantom, "\approx"] \\
\vdash \ulcorner e \urcorner : \|\tau\| \arrow[r, "\text{evaluates in } c \text{ steps to}", "\text{if } \epsilon = \mathbb{C} \text{ then } c \leq \kappa"'] & v_2
\end{tikzcd}$$

(a) First run

$$\begin{tikzcd}
\vdash^{\kappa}_{\mathbb{S}} e : \tau \arrow[r, "\text{evaluates to}"] \arrow[d, "\text{translates to}"] & v_1 \arrow[d, phantom, "\approx"] \\
\vdash \ulcorner e \urcorner : \|\tau\| \arrow[r, "\text{change propagates in } c \text{ steps to}", "c \leq \kappa"'] & v_2
\end{tikzcd}$$

(b) Incremental run under some input changes

Figure 5: Schematic representation of the basic properties of the translation

that application of destructors incurs unary cost while application of constructors incurs zero cost.

1.3 Overview of the contributions

The main contribution of this internship focuses on proving that the cost derived by a typing derivation in CostIt is sound with respect to a concrete change propagation semantics. To this end, we designed a target language, namely saML, that provides the infrastructure for incremental computation and we defined a translation that given a CostIt program, transforms it to a self-adjusting saML program. The soundness proof shows that the cost of change propagation in the saML program is less or equal to the cost derived by the type system.

The translation is type directed and it is defined by induction on the typing derivation. Similar to the typing rules, the translation scheme also has two modes, one for change propagation and one for from-scratch evaluation. During translation in change propagation mode a CostIt program is translated to a saML program that evaluates to the same result as the source program. Moreover, given a change in the inputs the target program can propagate changes to the output with cost less or equal to the cost derived by the type system. Change propagation in the target program yields the same result as complete re-evaluation of the source program under the same input changes. During translation in from-scratch mode, a CostIt program is transformed to a saML program that evaluates to the same result as the source program and its execution time is less or equal to the cost derived by the type system. Note that in this mode the derived program is not self-adjusting. The above are represented schematically in fig. 5.

The rest of the report is structured as follows: in section 2 we give an example of the translation, in section 3 we present the static and dynamic semantics of the target language, in section 4 we present the translation and the soundness results. Finally, in sections 5 and 6 we discuss related and future work.

2 Self-adjusting computation by example

Continuing the example of the higher-order function `map` we gave earlier, we will present the main concepts of our translation by explaining the translation of map.

Our goal is to translate `map` to a self-adjusting saML program that can update in place the parts of the output that depend on data that can change. Furthermore the translated function should not incur a change propagation cost that is greater than the cost in the cost annotation of its type. In order to be able to differentiate between values that can change and values that cannot and also be able to update changeable values we will enclose values that can change, i.e. values of type $(A)^{\mathbb{C}}$ for some A, in mutable references. Moreover, since in the list at most α elements can change and α has to be less or equal to the total number of elements in the list, we need a way to differentiate between elements in a list that can change and elements that cannot. We achieve this by translating the original list to a list of a sum type whose left variant will hold values that are stable and whose right variant values that can change. The type of `map` in the target language, that reflects the above modifications is

```
map f e  =  case_L e of
            | [] → []
            | h :: tl → f h :: map f tl
```

(a)

```
⌜map⌝ f e  =  case_L e of
              | [] → []
              | h :: tl →
                case h of
                | h_l → (inl !(f (ref h_l))) :: ⌜map⌝ f tl
                | h_r → let l = f h_r in
                        let () = push(l, λ(). f h_r) in
                        inr l :: ⌜map⌝ f tl
```

(b)

Figure 6: Mapping function in the source (a) and the target language (b).

$$(\texttt{ref int} \to \texttt{ref int}) \to \texttt{list}\ (\texttt{int} + \texttt{ref int}) \to \texttt{list}\ (\texttt{int} + \texttt{ref int})$$

Since the function passed as argument expects an argument of `ref int` whenever we hit an element that is stable, and thus of type `int`, we will have to put it in a reference cell before we apply the function to it. Furthermore, since the result will be stable, we will dereference the result of the application before putting it in the left variant of the sum type. Note that since the input of the function is stable and the body function is stable and also does not capture any variables that can change from the environment, the application will always yield the same result. When we hit an element that can change, we can immediately apply the function to it and wrap the result in the right variant of the type. Furthermore, since the input can change the result on the application needs to be reevaluated during change propagation. During the first run, we create a new location in which we store the result of the application, after dereferencing it. This is also the location that we put in the list computed by `map`. During change propagation we will recompute the application and we will store the new result, after dereferencing it, to the location we created during the first run. The list we computed during the first run will now contain the updated location.

In order to store the computations that need to be re-executed during change propagation and the locations that need to be updated, we modify the language runtime in order to maintain a global queue in which we push pairs of locations and closures. We enrich the language with new primitives in order to be able to manipulate the queue. We defer the presentation of the static and dynamic semantic of these new primitives untilsection 3. During change propagation, we pop an element from the queue, we evaluate the closure and we update in place the corresponding location with the result computed by the closure. We repeat until the queue is empty. In our example we push in the queue the application of the argument function to the head of the list, in order to re-execute during change propagation. The translated code is shown in fig. 6.

We can now show a glimpse of how the function will execute when applied to a list. Assume that we want to apply $\lambda x.\, x + 1$ to each element of a four element list, of which at most two can change. This function can be given the type $((\texttt{int})^{\mathbb{C}} \xrightarrow{\mathbb{C}(3)} (\texttt{int})^{\mathbb{C}})^{\square}$ and it is translated to $\lambda x.\, \texttt{ref}\ (!x + 1)$. Note that the function is executed from-scratch during change propagation and although addition incurs only a unit cost we also have to account for the cost of referencing and dereferencing that sums up to two additional units. There is no fundamental reason why we picked a function that executes from scratch, we could have considered a function that can change propagate but that would complicate our example. The partial application $\texttt{map}\ \lambda x.\, x + 1$ can now be given the type

$$\texttt{list}\,[4]^2\ (\texttt{int})^{\mathbb{C}} \xrightarrow{\mathbb{S}(3\cdot 2)} \texttt{list}\,[4]^2\ (\texttt{int})^{\mathbb{C}}$$

We want to make sure that the cost of change propagation will be at most six. Assume that we want to evaluate the expression $\texttt{map}\ (\lambda x.\, x + 1)\ [x_1,\ x_2,\ x_3,\ x_4]$ in the environment $x_1 : (\texttt{int})^{\mathbb{S}} \mapsto 1,\ x_2 :$

$(\mathtt{int})^{\mathbb{C}} \mapsto 42,\ x_3 : (\mathtt{int})^{\mathbb{S}} \mapsto 3,\ x_4 : (\mathtt{int})^{\mathbb{C}} \mapsto 4$. This would translate to evaluating the expression $\ulcorner\mathtt{map}\urcorner\ \lambda x.\, \mathtt{ref}\ (!x + 1)\ [\mathtt{inl}\ x_1,\ \mathtt{inr}\ x_2,\ \mathtt{inl}\ x_3,\ \mathtt{inr}\ x_4]$ in the environment $x_1 : \mathtt{int} \mapsto 1,\ x_2 : \mathtt{int} \mapsto l_1,\ x_3 : \mathtt{int} \mapsto 3,\ x_4 : \mathtt{int} \mapsto l_2$ and the store $l_1 : \mathtt{int} \mapsto 42,\ l_2 : \mathtt{int} \mapsto 4$.

After the first run, we obtain the store $l_1 : \mathtt{int} \mapsto 42,\ l_2 : \mathtt{int} \mapsto 4,\ l_1' : \mathtt{int} \mapsto 43,\ l_2' : \mathtt{int} \mapsto 5$ (we omit the locations that are not present in the input or the output). The output list is $[\mathtt{inl}\ 1,\ \mathtt{inr}\ l_1',\ \mathtt{inl}\ 4,\ \mathtt{inr}\ l_2']$ and the global queue is

$$[(l_1', \lambda().\, (\lambda x.\, \mathtt{ref}\ (!x + 1))\ l_1),\ (l_2', \lambda().\, (\lambda x.\, \mathtt{ref}\ (!x + 1))\ l_2]$$

Assume now that we update the input and want to recompute the result. The updated store is $l_1 : \mathtt{int} \mapsto 2,\ l_2 : \mathtt{int} \mapsto 4$. The change propagation mechanism will run the closures in the queue in the order that they were pushed and it will update the location in the first component of each pair with the value contained in the location that is returned from the closure. After running the closures and updating the locations the new store would be $l_1 : \mathtt{int} \mapsto 2,\ l_2 : \mathtt{int} \mapsto 4,\ l_1 : \mathtt{int} \mapsto 3,\ l_2 : \mathtt{int} \mapsto 5$. The references in the output list now point to locations that contain values that correspond to the updated result. Furthermore, the cost of pure computation (without taking into account the cost of queue operations or updating the references) is exactly 6.

3 Semantics of the target language

The target language is a simply typed lambda calculus with general references. The references are added to the target language in order to hold the values that can potentially change during subsequent runs. During change propagation only the value of the memory locations that the references point to will be recomputed and the references will be updated in place. Since the type system tracks the values that can potentially change between two runs and ensures that their types are annotated as changeable the translation scheme can ensure that every changeable component of the output will be stored in a memory location.

The language is also modified to add support for change propagation. The runtime of the language maintains a global queue that contains the computations that need to be re-executed during change propagation. The elements of the queue are pairs of a list of references and a function from unit to a list of references. During change propagation the recorded computation will be evaluated and will return a list of locations that will have the same length as the list in the first component and moreover elements in the same position in the two lists will have the same type. Note that this constraint is not enforced by the type system, but it will be an invariant of the translation. The values of the locations in the first list have to be updated with the values of the corresponding locations in the list that results from evaluation of the function.

We enrich the language with new primitives that allow us to interface with the queue. In order to be able to type these new primitives we need to be able to type lists that contain references of any type. Thus we need to enrich the language with a non-homogeneous list construct. Note that elimination of this new construct only happens during change propagation, which is a low level algorithm expressed outside the target language. Thus, we do not need to add semantics for the elimination of this new construct to the language. Modifying the language to also include destruction of non-homogeneous lists is not fundamentally hard but it would require enriching the type system with existential types.

In the remaining section we will present the semantics of the new primitives, that allow us to push to and empty the queue, and also the semantics of non-homogeneous lists that allow us to type the new primitives. The semantics of the rest of the language are fairly standard. The interested reader can refer to the technical appendix for a more detailed presentation.

3.1 Static and dynamic semantics

The semantics of non-homogeneous lists are largely similar to those of homogeneous lists. The only difference is that we allow inserting elements of arbitrary type and that the type of non-homogeneous list does not carry type information for the elements.

The primitive push expects a tuple of a non-homogeneous list and a function from unit to a non-homogeneous list, and pushes it to the global queue. The result of this computation is of unit type. The primitive drop takes as an argument an expression of an arbitrary type, empties the current queue, and returns the value that the given expression evaluates to. Note that during from-scratch evaluation we do not pop from the queue and so there is no need for a primitive to perform a popping operation. The typing rules for the new constructs are straight-forward and can be seen in fig. 7.

In fig. 8 we present the big-step dynamic semantics. The usual big-step semantics are modified to also return the queue and the cost of the computation. Unlike stores, queues are not threaded through the big-step rules. Instead, in order to derive the final queue, we append the queues that result from the premises of the rules. The evaluation is deterministic and the allocator when given a certain store will always return the same location, which should not be already present in the store.

$$\frac{}{\Gamma \vdash \texttt{nil} : \texttt{list}} \qquad \frac{\Gamma \vdash e : \tau \qquad \Gamma \vdash es : \texttt{list}}{\Gamma \vdash \texttt{cons}\ e\ es : \texttt{list}} \qquad \frac{\Gamma \vdash e : \tau}{\Gamma \vdash \texttt{drop}(e) : \tau} \qquad \frac{\Gamma \vdash e : \texttt{list} \qquad \Gamma \vdash f : \texttt{unit} \to \texttt{list}}{\Gamma \vdash \texttt{push}(e,\ f) : \texttt{unit}}$$

Figure 7: Typing rules for the target language

$$\frac{}{\texttt{nil},\ \sigma \Downarrow^L \texttt{nil},\ \sigma,\ \varnothing,\ 0} \qquad \frac{e,\ \sigma \Downarrow^L v,\ \sigma',\ Q_1,\ c_1 \qquad es,\ \sigma \Downarrow^L vs,\ \sigma',\ Q_2,\ c_2}{\texttt{cons}\ e\ es,\ \sigma \Downarrow^L \texttt{cons}\ v\ vs,\ \sigma'',\ Q_1 + Q_2,\ c_1 + c_2} \qquad \frac{e,\ \sigma \Downarrow^L v,\ \sigma',\ Q,\ c}{\texttt{drop}(e),\ \sigma \Downarrow^L v,\ \sigma',\ \varnothing,\ c}$$

$$\frac{e,\ \sigma \Downarrow^L v,\ \sigma',\ Q,\ c}{\texttt{push}(e,\ f),\ \sigma \Downarrow^L (),\ \sigma,\ Q + [(e,\ f)],\ c} \qquad \frac{e,\ \sigma \Downarrow^L v,\ \sigma',\ D,\ c \qquad l = \texttt{fresh}_L(\sigma')}{\texttt{ref}\ e,\ \sigma \Downarrow^L l,\ \sigma'[l \mapsto v],\ D,\ c + 1}$$

$$\frac{\begin{array}{c} e_1,\ \sigma \Downarrow^L \texttt{fix}\ f(x).\, e,\ \sigma',\ D_1,\ c_1 \\ e_2,\ \sigma' \Downarrow^L v',\ \sigma'',\ D_2,\ c_2 \qquad [x \mapsto v', f \mapsto \texttt{fix}\ f(x).\, e]e,\ \sigma'' \Downarrow^L \texttt{fix}\ f(x).\, e_1',\ \sigma''',\ D_3,\ c_3 \end{array}}{e_1\ e_2,\ \sigma \Downarrow^L v,\ \sigma''',\ D_1 + D_2 + D_3,\ c_1 + c_2 + c_3 + 1}$$

Figure 8: Big-step semantics of the target language

3.2 Change Propagation

The main idea behind the change propagation algorithm is to run each computation that has been pushed to the queue and update the corresponding locations with the new values, as described earlier in this section. Note that, unlike previous work, we do not construct a dependency graph that tracks which values depend on values that have been changed. Although that would allow us to only update the values that depend on values that have *actually* changed, this information is not statically known and thus the type system counts the cost of recomputing any value that can potentially change, regardless of whether the value actually changes or not. Therefore, an algorithm that recomputes all the values that can potentially change suffices to show that the cost incurred by the type system is sound.

For change propagation to be sound it is imperative that the locations on which the result of a deferred computation depends have already been updated before the computation gets evaluated. Our translation will have the invariant that if a pair has been pushed to the queue after another one, then it should be the case that all of the locations in the first component of the first pair were created after the second pair was pushed to the queue. Since the result of a computation cannot depend on a location that has not been created yet, it should be the case that no locations that affect the result of a deferred computation are updated after the computation gets evaluated. Since all of the locations that are changeable will get updated eventually, it should also be the case that all the locations on which a computation depends will have been updated before the latter is evaluated.

To avoid dealing with location updates in the proofs, instead of updating the value of the already existing locations we create freshly allocated locations that hold the updated value and we maintain a partial bijection that maps the locations holding the old values to the corresponding new ones. The deferred computation will be responsible for returning a list of freshly allocated locations that hold the new values of the corresponding old locations. To get the updated result it suffices to apply the partial bijection to it in order to rename the old locations to the new ones.

The change propagation algorithm works as follows. Assume that the initial store is σ_i and after the first run we obtain a value v and a store σ_f. Let σ_c be the updated store and $\mathtt{dom}(\sigma_i) \cap \mathtt{dom}(\sigma_c) = \emptyset$. Let also β be a partial bijection that maps locations from the initial store to the locations of the updated store that hold the updated value. If a location does not belong to the domain of the bijection then its value does not change. We give as input to the change propagation algorithm the queue, the updated store and the store that we obtain after the first run. In the case that the domain of the bijection is empty then no update has happened so there is no need to reevaluate the deferred computations. In the presence of updates, the algorithm will pop a pair from the queue and will evaluate the closure after applying the partial bijection to it in order to rename the locations that appear free and whose value has changed. Since some locations that appear free may not have changed, and thus may not have a mapping in the updated store, the closure is evaluated in the updated store extended with the store that is obtained from first run. After the evaluation we set the updated store to be the store that results from the evaluation of the closure after σ_f is removed. Note that in the absence of store updates, the store that results from the evaluation is always an extension of the input store. The bijection is extended to map each of the locations in first component of the pair to the corresponding locations in the list that results from the evaluation of the closure. In the case that the two lists are of different length or the new bijection is not well-defined the execution of change propagation will result in a run-time error. Such erroneous programs however will never arise from the translation, as shown by our soundness theorem. The change propagation algorithm can be seen in algorithm 1.

Algorithm 1 $Q,\ \sigma_c,\ \sigma_f,\ \beta \rightsquigarrow_L \sigma'_f,\ \beta',\ c$

```
if $\beta = \emptyset \wedge \sigma_c = \emptyset$ then
    return $(\sigma_f, \beta, 0)$
else
    $\sigma'_f \leftarrow \sigma_c$
    $\beta' \leftarrow \beta$
    while $Q \neq \emptyset$ do
        $(\vec{l},\ f) \leftarrow \mathtt{pop}(Q)$
        if $\beta'(f)\ (),\ \sigma'_f + \sigma_f \Downarrow^L \vec{l'},\ \sigma''_f,\ \varnothing,\ c'$ then
            $\sigma'_f \leftarrow \sigma''_f \setminus \sigma_f$
            $\beta' \leftarrow (\beta' \otimes \vec{l} \mapsto \vec{l'})$
            $c \leftarrow c + c'$
        else
            $\mathtt{error}()$
        end if
    end while
    return $(\sigma'_f,\ \beta',\ c)$
end if
```

4 Translation

The translation transforms a source program into a target program that is capable of propagating changes in the inputs (free variables) to the output. The target program stores each changeable value in a reference cell and during change propagation only the values of memory locations need to be updated. The translation takes care of creating new memory locations to store values that need to be updated and pushing computations that need to be reevaluated during change propagation in the queue.

4.1 Translating Types

An invariant that should be maintained by the translation is that the target program should be typable with a target type that corresponds to the type of the source program. This correspondence is obtained by translating the source language types to target language types. The main idea behind translating types is to translate a changeable type to a mutable reference. Moreover, in order to distinguish between elements of a list that can change and elements that cannot, we translate lists of type τ to lists of type $\|\tau^{\downarrow\Box}\| + \|\tau\|$, where $\|\cdot\|$ is the translation function. When an element of the list is of the left variant then its value is stable and its type is stabilized in order to reflect this. When the element is of the right variant then the value can be changeable. Another point that needs careful handling is universal quantifiers, as their introduction rules in the type system incur zero cost in both change propagation and from scratch cost. To reflect this in the target language, for all quantified types are translated to functions of a unit type argument. For the rest of the types the translation is defined as a homomorphic erasure of the indexes.

$$
\begin{array}{lcl}
\|\texttt{int}\|^A & = & \texttt{int} \\
\|\tau_1 + \tau_2\|^A & = & \|\tau_1\| + \|\tau_2\| \\
\|\tau_1 \xrightarrow{\mu(\kappa)} \tau_2\|^A & = & \|\tau_1\| \to \|\tau_2\| \\
\|\forall i \overset{\mu(\kappa)}{::} S.\ \tau\|^A & = & \texttt{unit} \to \|\tau\|
\end{array}
\qquad
\begin{array}{lcll}
\|(A)^{\mathbb{C}}\| & = & \texttt{ref}\ \|A\|^A & \\
\|(A)^{\mu}\| & = & \|A\|^A & \text{if } \mu = \mathbb{S} \vee \mu = \Box \\
\|\texttt{unit}\| & = & \texttt{unit} & \\
\|\tau_1 \times \tau_2\| & = & \|\tau_1\| \times \|\tau_2\| & \\
\|\texttt{list}\,[n]^{\alpha}\ \tau\| & = & \texttt{list}\ \|\tau\| + \|\tau^{\downarrow\Box}\| & \\
\|\exists i.\ \tau\| & = & \|\tau\| & \\
\|C \to \tau\| & = & \|\tau\| & \\
\|C \wedge \tau\| & = & \|\tau\| &
\end{array}
$$

Figure 9: Translation of types

4.2 Translating Expressions

The translation of the expressions is defined by induction on the typing derivation. The translation should enforce that a changeable value is stored in a mutable reference and that the closure that recomputes its value is pushed to the queue. In this subsection we will discuss and present the most interesting cases of the translation. For a full presentation of the rules the reader can refer to the technical appendix. Selected rules are presented in fig. 28.

For constructors of pair and sum type construction, rules **inl** and **pair**, the translation is homomorphically defined using the translation of the premises. The derived cost is the sum of the cost annotations of the premises. For pair destruction, that corresponds to rule **fst**, in from-scratch mode, we count the cost that is obtained by the premises and we add a unit to it that corresponds to the application of the destructor. In change propagation mode we only need to count the cost of evaluating the premises as the result is updated in place and the constructor will not be reapplied. The translation of sum type destruction when the scrutinee is stable, rule **case1**, is straight forward and the cost is derived using the same principles as in pair destruction. When the scrutinee is changeable the control flow may follow a different path in a subsequent run for which there are no recorded computations and thus the result cannot be updated in place by propagating changes. Therefore, in change propagation mode, the result of the expression should be recomputed from-scratch in each subsequent run. During the first run the result of the expression is computed and each changeable component is placed in freshly allocated references. Subsequently, the closure that computes the output of the application and creates new locations for the changeable components will be pushed to the queue along with the list of locations that where previously allocated. During change propagation the closure will be evaluated and the old locations will be mapped to the corresponding new ones. This is illustrated in the rule **case2**. The meta-function $\underline{\mathbf{deep}}(\cdot,\ \cdot)$ is responsible for generating code that stores the changeable parts of the given expression to the new locations and return a reconstructed expression that contains the new locations and a (non-homogeneous) list consisting of these locations. The meta-function $\underline{\mathbf{deep'}}(\cdot,\ \cdot)$ is similar with the difference that it only returns a location list and does not reconstruct the expression. The cost derived by the type system in this case is the cost of change propagating changes in the scrutiny, the cost of evaluating from-scratch the path that is taken, the cost associated with creating new references, and also the three additional units that correspond to dereferencing the location that holds the value of the scrutinee, the destructor application and the application of the closure. When the scrutinee is changeable and the typing is in from-scratch mode then the translation follows the same principles with the case when the scrutinee is stable, with the only difference that before case analyzing the scrutinee the translation should dereference it and the derived cost should be modified to reflect this.

The translation of function abstraction is defined as the abstraction of the translation of the function body. The change propagation (resp. from-scratch evaluation) cost in the function type is the cost that is obtained by the typing of its body in change propagation (resp. from-scratch evaluation) mode.

For function application we should again consider the cases where the function applied is in change propagation or from scratch mode separately. We should also distinguish between the cases where the function is itself changeable or not. In change propagation mode, when a change propagation mode function that itself is not changeable is applied to an argument, it is translated to the application of the translation of the function to the translation of the argument. This case corresponds to the rule **app1**. Since the function is in change propagation mode its translated code will propagate potential changes to the argument and to the variables that are captured in its body to the result. The derived cost is the sum of the costs that are obtained by the premises plus the cost annotation in the function type, that is the upper bound on the cost of propagating changes through the function body. When a from-scratch mode function that is not itself changeable is applied to an argument then the result should be recomputed during change propagation. The translation will evaluate the function and the argument and then it will apply the former to the latter creating fresh locations to store the changeable components of the result. This is illustrated in the rule **app2**. The cost derived by the type system in this case is the cost of change propagating changes in the function and in the argument, the cost of applying the function, and also the cost associated with creating new references. The three additional units represent the cost of dereferencing and applying the function, and applying the recorded closure. In the case that the function is itself changeable, that corresponds to the rule **app3**, the only difference to the above is that in order to apply the function the reference that stores it is dereferenced.

Function application in from scratch mode translates to function application of the translation of the function to the translation of the argument. In the case that the function is changeable its translation is dereferenced before applied. The derived cost is the cost of evaluating the function, the cost of evaluating the argument, the cost of the function application and the cost of dereferencing the function if it is changeable.

The translation of primitive function application is similar to the translation of from-scratch mode function application, as primitive functions are by default evaluated from-scratch. Since primitive functions are closed when the input is stable the output will also be stable. When one of the inputs is changeable then the result is also changeable and the whole expression is re-executed during change propagation.

The translation of list construction is straight-forward. When a stable element is cons-ed in the list then its translation is wrapped in the constructor of the left variant of the sum type and it is cons-ed with the translation of the rest of the list. If a changeable element is inserted in the list then its translation is wrapped in the constructor of the right variant. Destruction of lists translates to destruction of the translation of the scrutiny. In the cons case the head should be destructed further in order to determine if it is stable or changeable. If the head is stable (resp. changeable) then the translation that corresponds to the typing of stable (resp. changeable) case is returned.

In the **nochange** case the expression is typeable in a stable context and thus itself cannot change, as explained in the introduction. The translation will be the translation of the expression with the addition that the computation accumulated in the queue during its evaluation will be discarded and the result will be converted in order to have a stable type. Since no change propagation will happen, in the change propagation mode the cost is zero. In the from-scratch mode the cost is the cost of evaluating the expression plus the unary cost induced by the queue discard.

Lastly, we discuss the translation of expressions that are typed using the subtyping rule. Given an expression of the target language whose type is the translation of the subtype, we need to convert it to an expression whose type is the translation of the supertype. To this end, we define a coercion meta-function that given a target expression of the subtype will generate code that converts it to an expression of the supertype. This meta-function is defined by induction on the subtyping derivation. The generated code may perform operations that incur an extra from scratch evaluation cost. Correspondingly, the code may push extra computations to the queue that incur an extra change propagation cost. The cost that is derived by the subtyping derivation reflects the extra cost that is incurred by the code generated by the coercion. The definition of the coercion meta-function for selected cases can be seen in fig. 11.

4.3 Soundness

In this section we will present the main soundness properties of the translation scheme. More specifically, we will prove that the translated program is a well typed program and that evaluation and change propagation in the translated program produce results that are correct with respect to the semantics of the source program. Furthermore, we will prove that the cost incurred by evaluation and change propagation are less or equal to the cost derived by the type system in from-scratch and change propagation mode respectively.

Our first theorem states that given a well-typed CostIt program its translation exists and furthermore it is a well-typed program in the target language.

Theorem 1 (Totality of the translation and type soundness)
Let $\Delta; \Phi; \Gamma \vdash_{\epsilon}^{\kappa} e : \tau$. Then $\Delta; \Phi; \Gamma \vdash_{\epsilon}^{\kappa} e : \tau \hookrightarrow \ulcorner e \urcorner$ and $\|\Gamma\| \vdash \ulcorner e \urcorner : \|\tau\|$

Proof. The proof is straight-forward and proceeds by induction on the typing derivation of the source program. Note that in certain cases we need to prove the type soundness of the meta-functions used by the translation. □

Before proceeding to stating and proving the soundness of the translation and the type system we should define a notion of *similarity* between source and target values. In order to prove correctness of the evaluation of the translated program we will define a relation that relates a source value and a target value that are similar under a store. Two expressions are similar if they evaluate to similar values. In order to prove correctness of change propagation, we will define a *two-way-similarity* relation that relates two source expressions, one that represents the initial expression and a second one that represents the updated expression and a target expression, under two stores – the initial and the updated one – and a bijection between their domains. The three expressions are related if the target expression is similar under the first store with the first source expression and moreover if the target expression, after applying the partial bijection to its free locations, is similar under the combination of the two stores to the second source expression. Furthermore for values that are not allowed to change, i.e. they have a stable type, all the three expressions must be similar. Two expressions are two-way-similar if they evaluate to two-way-similar values. We define the two relations by induction on the type of the values. Their definitions can be seen in fig. 12 and fig. 13.

We can now state the soundness theorems. Our first theorem states the soundness of the translation and cost derived by the type system in from-scratch mode. To ease readability we state the theorem when there is only one free variable in the environment but it can be generalized for any number of free variables.

Theorem 2 (Type and translation soundness, $\mathbb{C}$ mode)
Assume that $\cdot; \cdot; x : \tau' \vdash_{\mathbb{C}}^{\kappa} e : \tau \hookrightarrow \ulcorner e \urcorner$ and $v_s \approx_{\sigma}^{\tau} v_t$. Then there exist v_s', v_t', σ', j and c, such that (1) $[x \mapsto v_s]e \Downarrow v_s',\ j$ (2) $[x \mapsto v_t]\ulcorner e \urcorner,\ \sigma \Downarrow v_t',\ \sigma',\ \varnothing,\ c$, (3) $\models c \stackrel{.}{\leq} \kappa$ and (4) $v_s' \approx_{\sigma'}^{\tau} v_t'$.

$$\frac{\Delta;\Phi;\Gamma \vdash^{\kappa_1}_{\epsilon} e_1 : \tau_1 \hookrightarrow \ulcorner e_1 \urcorner \qquad \Delta;\Phi;\Gamma \vdash^{\kappa_2}_{\epsilon} e_1 : \tau_2 \hookrightarrow \ulcorner e_2 \urcorner \qquad \Delta;\Phi \models \kappa \doteq \kappa_1 \mathrel{\hat{+}} \kappa_2}{\Delta;\Phi;\Gamma \vdash^{\kappa}_{\epsilon} (e_1, e_2) : \tau_1 \times \tau_2 \hookrightarrow (\ulcorner e_1 \urcorner, \ulcorner e_2 \urcorner)}\ \textbf{pair}$$

$$\frac{\Delta;\Phi;\Gamma \vdash^{\kappa}_{\epsilon} e : \tau_1 \hookrightarrow \ulcorner e \urcorner}{\Delta;\Phi;\Gamma \vdash^{\kappa}_{\epsilon} \mathtt{inl}\ e : (\tau_1 + \tau_2)^{\mathbb{S}} \hookrightarrow \mathtt{inl}\ \ulcorner e \urcorner}\ \textbf{inl} \qquad \frac{\Delta;\Phi;\Gamma \vdash^{\kappa'}_{\epsilon} e : \tau_1 \times \tau_2 \hookrightarrow \ulcorner e \urcorner \qquad \Delta;\Phi \models \kappa \doteq (\kappa' \mathrel{\hat{+}} \epsilon = \mathbb{C}\ ?\ 1\ :\ 0)}{\Delta;\Phi;\Gamma \vdash^{\kappa}_{\epsilon} \mathtt{fst}\ e : \tau_1 \hookrightarrow \mathtt{fst}\ \ulcorner e \urcorner}\ \textbf{fst}$$

$$\frac{\Delta;\Phi;\Gamma, f : (\tau_1 \xrightarrow{\delta(\kappa)} \tau_2)^{\mathbb{S}}, x : \tau_1 \vdash^{\kappa}_{\delta} e : \tau_2 \hookrightarrow \ulcorner e \urcorner}{\Delta;\Phi;\Gamma \vdash^{0}_{\epsilon} \mathtt{fix}\ f(x).\, e : (\tau_1 \xrightarrow{\delta(\kappa)} \tau_2)^{\mathbb{S}} \hookrightarrow \mathtt{fix}\ f(x).\ulcorner e \urcorner}\ \textbf{fix1}$$

$$\frac{\Delta;\Phi;\Gamma \vdash^{\kappa_1}_{\mathbb{S}} e_1 : (\tau_1 \xrightarrow{\mathbb{S}(\kappa')} \tau_2)^{\mu} \hookrightarrow \ulcorner e_1 \urcorner \qquad \Delta;\Phi;\Gamma \vdash^{\kappa_2}_{\mathbb{S}} e_2 : \tau_1 \hookrightarrow \ulcorner e_2 \urcorner \qquad \mu \leq \mathbb{S} \qquad \Delta;\Phi \models \kappa \doteq \kappa' \mathrel{\hat{+}} \kappa_1 \mathrel{\hat{+}} \kappa_2}{\Delta;\Phi;\Gamma \vdash^{\kappa}_{\mathbb{S}} e_1\ e_2 : \tau_2 \hookrightarrow \ulcorner e_1 \urcorner\, \ulcorner e_2 \urcorner}\ \textbf{app1}$$

$$\frac{\begin{array}{c}\Delta;\Phi;\Gamma \vdash^{\kappa_1}_{\mathbb{S}} e_1 : (\tau_1 \xrightarrow{\mathbb{C}(\kappa')} \tau_2)^{\mathbb{C}} \hookrightarrow \ulcorner e_1 \urcorner \\ \Delta;\Phi;\Gamma \vdash^{\kappa_2}_{\mathbb{S}} e_2 : \tau_1 \hookrightarrow \ulcorner e_2 \urcorner \qquad \mathbb{C} \trianglelefteq \tau_2 \qquad \Delta;\Phi \models \kappa \doteq \kappa' \mathrel{\hat{+}} \kappa_1 \mathrel{\hat{+}} \kappa_2 \mathrel{\hat{+}} \mathtt{cost}_{\mathtt{deepref}}(\tau_2) \mathrel{\hat{+}} 3\end{array}}{\Delta;\Phi;\Gamma \vdash^{\kappa}_{\mathbb{S}} e_1\ e_2 : \tau_2 \hookrightarrow \begin{array}{l}\mathtt{let}\ l\ =\ \ulcorner e_1 \urcorner\ \mathtt{in}\ \mathtt{let}\ x\ =\ \ulcorner e_2 \urcorner\ \mathtt{in}\ \mathtt{let}\ r\ =\ \underline{\mathbf{deep}}(!l\ x,\ \tau_2)\ \mathtt{in} \\ \mathtt{let}\ ()\ =\ \mathtt{push}(\mathtt{snd}\ r,\ \lambda().\, \underline{\mathbf{deep}'}(!l\ x,\ \tau_2))\ \mathtt{in}\ \mathtt{fst}\ r\end{array}}\ \textbf{app2}$$

$$\frac{\Delta;\Phi;\Gamma \vdash^{\kappa_1}_{\mathbb{C}} e_1 : (\tau_1 \xrightarrow{\mathbb{C}(\kappa')} \tau_2)^{\mathbb{S}} \hookrightarrow \ulcorner e_1 \urcorner \qquad \Delta;\Phi;\Gamma \vdash^{\kappa_2}_{\mathbb{C}} e_2 : \tau_1 \hookrightarrow \ulcorner e_2 \urcorner \qquad \mathbb{C} \trianglelefteq \tau_2 \qquad \Delta;\Phi \models \kappa \doteq \kappa' \mathrel{\hat{+}} \kappa_1 \mathrel{\hat{+}} \kappa_2 \mathrel{\hat{+}} 1}{\Delta;\Phi;\Gamma \vdash^{\kappa}_{\mathbb{C}} e_1\ e_2 : \tau_2 \hookrightarrow \ulcorner e_1 \urcorner\, \ulcorner e_2 \urcorner}\ \textbf{app5}$$

$$\frac{\begin{array}{c}\Delta;\Phi;\Gamma \vdash^{\kappa_e}_{\epsilon} e : (\tau_1 + \tau_2)^{\mu} \hookrightarrow \ulcorner e \urcorner \qquad \Delta;\Phi;\Gamma, x : \tau_1 \vdash^{\kappa'}_{\epsilon} e_1 : \tau \hookrightarrow \ulcorner e_1 \urcorner \\ \Delta;\Phi;\Gamma, y : \tau_2 \vdash^{\kappa'}_{\epsilon} e_2 : \tau \hookrightarrow \ulcorner e_2 \urcorner \qquad \mu \leq \mathbb{S} \qquad \Delta;\Phi \models \kappa \doteq \kappa_e \mathrel{\hat{+}} \kappa' \mathrel{\hat{+}} ((\epsilon = \mathbb{C})\ ?\ 1\ :\ 0)\end{array}}{\Delta;\Phi;\Gamma \vdash^{\kappa}_{\epsilon} \mathtt{case}(e, x.e_1, y.e_2) : \tau \hookrightarrow \mathtt{case}(\ulcorner e \urcorner, x.\ulcorner e_1 \urcorner, y.\ulcorner e_2 \urcorner)}\ \textbf{case1}$$

$$\frac{\begin{array}{c}\Delta;\Phi;\Gamma \vdash^{\kappa_e}_{\mathbb{S}} e : (\tau_1 + \tau_2)^{\mathbb{C}} \hookrightarrow \ulcorner e \urcorner \qquad \Delta;\Phi;\Gamma, x : \tau_1 \vdash^{\kappa'}_{\mathbb{C}} e_1 : \tau \hookrightarrow \ulcorner e_1 \urcorner \\ \Delta;\Phi;\Gamma, y : \tau_2 \vdash^{\kappa'}_{\mathbb{C}} e_2 : \tau \hookrightarrow \ulcorner e_2 \urcorner \qquad \mathbb{C} \trianglelefteq \tau \qquad \Delta;\Phi \models \kappa \doteq \kappa_e \mathrel{\hat{+}} \kappa' \mathrel{\hat{+}} \mathtt{cost}_{\mathtt{deepref}}(\tau) \mathrel{\hat{+}} 3\end{array}}{\Delta;\Phi;\Gamma \vdash^{\kappa}_{\mathbb{S}} \mathtt{case}(e, x.e_1, y.e_2) : \tau \hookrightarrow \begin{array}{l}\mathtt{let}\ l\ =\ \ulcorner e \urcorner\ \mathtt{in}\ \mathtt{let}\ r\ =\ \underline{\mathbf{deep}}(\mathtt{case}(!l, x.\ulcorner e_1 \urcorner, y.\ulcorner e_2 \urcorner),\ \tau)\ \mathtt{in} \\ \mathtt{let}\ ()\ =\ \mathtt{push}(\mathtt{snd}\ r,\ \lambda().\, \underline{\mathbf{deep}'}(\mathtt{case}(!l, x.\ulcorner e_1 \urcorner, y.\ulcorner e_2 \urcorner),\ \tau))\ \mathtt{in} \\ \mathtt{fst}\ r\end{array}}\ \textbf{case2}$$

$$\frac{\begin{array}{c}\Delta;\Phi;\Gamma \vdash^{\kappa_e}_{\epsilon} e : (\tau_1 + \tau_2)^{\mathbb{C}} \hookrightarrow \ulcorner e \urcorner \\ \Delta;\Phi;\Gamma, x : \tau_1 \vdash^{\kappa'}_{\mathbb{C}} e_1 : \tau \hookrightarrow \ulcorner e_1 \urcorner \qquad \Delta;\Phi;\Gamma, y : \tau_2 \vdash^{\kappa'}_{\mathbb{C}} e_2 : \tau \hookrightarrow \ulcorner e_2 \urcorner \qquad \mu \trianglelefteq \tau \qquad \Delta;\Phi \models \kappa \doteq \kappa_e \mathrel{\hat{+}} \kappa' \mathrel{\hat{+}} 2\end{array}}{\Delta;\Phi;\Gamma \vdash^{\kappa}_{\mathbb{C}} \mathtt{case}(e, x.e_1, y.e_2) : \tau \hookrightarrow \mathtt{case}(!\ulcorner e \urcorner, x.\ulcorner e_1 \urcorner, y.\ulcorner e_2 \urcorner)}\ \textbf{case3}$$

$$\frac{\Delta;\Phi;\Gamma \vdash^{\kappa_1}_{\epsilon} e : \tau' \hookrightarrow \ulcorner e \urcorner \qquad \Delta;\Phi \vdash^{\kappa_2}_{\epsilon} \tau' \sqsubseteq \tau \hookrightarrow \underline{\mathbf{g}} \qquad \Delta;\Phi \models \kappa_1 \mathrel{\hat{+}} \kappa_2 \mathrel{\dot{\leq}} \kappa}{\Delta;\Phi;\Gamma \vdash^{\kappa}_{\epsilon} e : \tau \hookrightarrow \underline{\mathbf{g}}(\ulcorner e \urcorner)}\ \sqsubseteq$$

$$\frac{\Delta;\Phi;\Gamma \vdash^{\kappa'}_{\epsilon} e : \tau \hookrightarrow \ulcorner e \urcorner \qquad \forall x \in \Gamma\ \ \Delta;\Phi \vdash \Gamma(x) \sqsubseteq \Gamma(x)^{\downarrow\Box} \qquad \Delta;\Phi \models \kappa \doteq (\epsilon = \mathbb{S}\ ?\ 0\ :\ \kappa')}{\Delta;\Phi;\Gamma,\Gamma' \vdash^{\kappa}_{\epsilon} e : \tau^{\downarrow\Box} \hookrightarrow \mathtt{drop}(\underline{\mathbf{conv}}(\ulcorner e \urcorner,\ \tau,\ \tau^{\downarrow\Box}))}\ \textbf{nochange}$$

$$\frac{\Delta;\Phi;\Gamma \vdash^{\kappa_1}_{\epsilon} e_1 : \tau^{\downarrow\Box} \hookrightarrow \ulcorner e_1 \urcorner \qquad \Delta;\Phi;\Gamma \vdash^{\kappa_2}_{\epsilon} e_2 : \mathtt{list}\,[n]^{\alpha}\ \tau \hookrightarrow \ulcorner e_2 \urcorner \qquad \Delta;\Phi \models \kappa \doteq \kappa_1 \mathrel{\hat{+}} \kappa_2}{\Delta;\Phi;\Gamma \vdash^{\kappa}_{\epsilon} e_1 :: e_2 : \mathtt{list}\left[n \mathrel{\hat{+}} 1\right]^{\alpha}\ \tau \hookrightarrow \mathtt{inl}\ \ulcorner e_1 \urcorner :: \ulcorner e_2 \urcorner}\ \textbf{cons1}$$

$$\frac{\Delta;\Phi;\Gamma \vdash^{\kappa_1}_{\epsilon} e_1 : \tau \hookrightarrow \ulcorner e_1 \urcorner \qquad \Delta;\Phi;\Gamma \vdash^{\kappa_2}_{\epsilon} e_2 : \mathtt{list}\,[n]^{\alpha-1}\ \tau \hookrightarrow \ulcorner e_2 \urcorner \qquad \Delta;\Phi \models \alpha \mathrel{\dot{>}} 0 \qquad \Delta;\Phi \models \kappa \doteq \kappa_1 \mathrel{\hat{+}} \kappa_2}{\Delta;\Phi;\Gamma \vdash^{\kappa}_{\epsilon} e_1 :: e_2 : \mathtt{list}\left[n \mathrel{\hat{+}} 1\right]^{\alpha}\ \tau \hookrightarrow \mathtt{inr}\ \ulcorner e_1 \urcorner :: \ulcorner e_2 \urcorner}\ \textbf{cons2}$$

$$\frac{\begin{array}{c}\Delta;\Phi;\Gamma \vdash^{\kappa_e}_{\epsilon} e : \mathtt{list}\,[n]^{\alpha}\ \tau \hookrightarrow \ulcorner e \urcorner \\ \Delta;\Phi \wedge n\ \doteq\ 0;\Gamma \vdash^{\kappa'}_{\epsilon} e_1 : \tau' \hookrightarrow \ulcorner e_1 \urcorner \qquad \Delta, i :: \iota;\Phi \wedge n \doteq i+1;\Gamma, h : \tau^{\downarrow\Box}, tl : \mathtt{list}\,[i]^{\alpha}\ \tau \vdash^{\kappa'}_{\epsilon} e_2 : \tau' \hookrightarrow \ulcorner e_2 \urcorner_l \\ \Delta, i :: \iota, \beta :: \iota;\Phi \wedge n \doteq i+1 \wedge \alpha \doteq \beta + 1;\Gamma, h : \tau, tl : \mathtt{list}\,[i]^{\beta}\ \tau \vdash^{\kappa'}_{\epsilon} e_2 : \tau' \hookrightarrow \ulcorner e_2 \urcorner_r \\ \Delta;\Phi \models \kappa \doteq \kappa_e \mathrel{\hat{+}} \kappa' \mathrel{\hat{+}} (\epsilon = \mathbb{C})\ ?\ 1\ :\ \ 0\end{array}}{\Delta;\Phi;\Gamma \vdash^{\kappa}_{\epsilon} \mathtt{case}_{\mathtt{L}}\ e\ \mathtt{of}\ \mathtt{nil}\ \rightarrow\ e_1 \mid \mathtt{cons}(h, tl)\ \rightarrow\ e_2 : \tau' \hookrightarrow \begin{array}{l}\mathtt{case}_{\mathtt{L}}\ \ulcorner e \urcorner\ \mathtt{of} \\ \mid\ \mathtt{nil}\ \rightarrow\ \ulcorner e_1 \urcorner \\ \mid\ \mathtt{cons}(s, tl)\ \rightarrow\ \mathtt{case}(s, h.\ulcorner e_2 \urcorner_l, h.\ulcorner e_2 \urcorner_r)\end{array}}\ \textbf{caseL}$$

Figure 10: Translation rules

$$\frac{\Delta;\Phi \models \kappa \mathbin{\hat{+}} \mathtt{cost}_{\mathtt{conv}}(\tau_1^{\downarrow\Box},\tau_1) \mathbin{\hat{+}} (\delta = \mathbb{C})\ ?\ (1 \mathbin{\hat{+}} \mathtt{cost}_{\mathtt{conv}}(\tau_2,\tau_2^{\downarrow\Box}))\ :\ 0 \mathrel{\dot{\leq}} \kappa'}{\Delta;\Phi \vdash_{\epsilon}^{0} (\tau_1 \xrightarrow{\delta(\kappa)} \tau_2)^{\Box} \sqsubseteq (\tau_1^{\downarrow\Box} \xrightarrow{\delta(\kappa')} \tau_2^{\downarrow\Box})^{\Box} \hookrightarrow \underline{\lambda} e.\lambda x.\, \underline{\mathbf{conv}}(e\ \underline{\mathbf{conv}}(x,\ \tau_1^{\downarrow\Box},\ \tau_1),\ \tau_2,\ \tau_2^{\downarrow\Box})} \rightarrow 1$$

$$\frac{\Delta;\Phi \vdash_{\delta}^{\kappa_1} \tau_1' \sqsubseteq \tau_1 \hookrightarrow \underline{\mathbf{g_1}} \qquad \Delta;\Phi \vdash_{\delta}^{\kappa_2} \tau_2 \sqsubseteq \tau_2' \hookrightarrow \underline{\mathbf{g_2}} \qquad \Delta;\Phi \models \kappa \mathbin{\hat{+}} \kappa_1 \mathbin{\hat{+}} \kappa_2 \mathbin{\hat{+}} (\delta = \mathbb{C})\ ?\ 1\ :\ 0 \leq \kappa'}{\Delta;\Phi_A \vdash_{\epsilon}^{0} \tau_1 \xrightarrow{\delta(\kappa)} \tau_2 \sqsubseteq \tau_1' \xrightarrow{\delta(\kappa')} \tau_2' \hookrightarrow \underline{\lambda} e.\mathtt{fix}\ f(x).\, \underline{\mathbf{g_2}}(e\ \underline{\mathbf{g_1}}(x))} \rightarrow 2$$

$$\frac{\Delta;\Phi \models \kappa \mathrel{\dot{=}} (\epsilon = \mathbb{C})\ ?\ 1 \mathbin{\hat{+}} \mathtt{m\hat{a}x}(\mathtt{cost}_{\mathtt{conv}}(\tau_1,\tau_1^{\downarrow\Box}),\mathtt{cost}_{\mathtt{conv}}(\tau_2,\tau_2^{\downarrow\Box}))\ :\ 0}{\Delta;\Phi \vdash_{\epsilon}^{\kappa} (\tau_1 + \tau_2)^{\Box} \sqsubseteq (\tau_1^{\downarrow\Box} + \tau_2^{\downarrow\Box})^{\Box} \hookrightarrow \underline{\lambda} e.\mathtt{case}(e, x.\mathtt{inl}\ \underline{\mathbf{conv}}(x,\ \tau_1,\ \tau_1^{\downarrow\Box}), y.\mathtt{inr}\ \underline{\mathbf{conv}}(y,\ \tau_2,\ \tau_2^{\downarrow\Box}))} \ \mathbf{+1}$$

$$\frac{\Delta;\Phi \models \alpha\ \dot{=}\ 0}{\Delta;\Phi \vdash_{\epsilon}^{0} \mathtt{list}\,[n]^{\alpha}\ \tau \sqsubseteq \mathtt{list}\,[n]^{\alpha}\ \tau^{\downarrow\Box} \hookrightarrow \underline{\lambda} e.e} \ \mathbf{l1}$$

$$\frac{\Delta;\Phi \models n \mathrel{\dot{=}} n' \qquad \Delta;\Phi \models \alpha \mathrel{\dot{=}} \alpha' \qquad \Delta;\Phi \vdash_{\mathbb{S}}^{\kappa'} \tau \sqsubseteq \tau' \hookrightarrow \underline{\mathbf{g}} \qquad \Delta;\Phi \models \kappa \mathrel{\dot{=}} \alpha \mathbin{\hat{\cdot}} \kappa'}{\Delta;\Phi \vdash_{\mathbb{S}}^{\kappa} \mathtt{list}\,[n]^{\alpha}\ \tau \sqsubseteq \mathtt{list}\,[n']^{\alpha'}\ \tau' \hookrightarrow \underline{\lambda} e.\mathtt{map}\ \lambda z.\ \mathtt{case}(z, x.\mathtt{inl}\ \underline{\mathbf{conv}}(\mathtt{drop}(\underline{\mathbf{g}}(\underline{\mathbf{conv}}(x,\ \tau^{\downarrow\Box},\ \tau))),\ \tau',\ \tau'^{\downarrow\Box}), y.\mathtt{inr}\ \underline{\mathbf{g}}\ y)\ e} \ \mathbf{l2\text{-}}\mathbb{S}$$

$$\frac{\begin{array}{c}\Delta;\Phi \models n \mathrel{\dot{=}} n' \qquad \Delta;\Phi \models \alpha \mathrel{\dot{=}} \alpha' \\ \Delta;\Phi \vdash_{\mathbb{C}}^{\kappa'} \tau \sqsubseteq \tau' \hookrightarrow \underline{\mathbf{g}} \qquad \Delta;\Phi \models \kappa \mathrel{\dot{=}} (n \mathbin{\hat{\cdot}} \kappa' \mathbin{\hat{+}} (n \mathbin{\hat{-}} \alpha) \mathbin{\hat{\cdot}} (\mathtt{cost}_{\mathtt{conv}}(\tau^{\downarrow\Box},\tau) \mathbin{\hat{+}} \mathtt{cost}_{\mathtt{conv}}(\tau',\tau'^{\downarrow\Box}) \mathbin{\hat{+}} 1))\end{array}}{\Delta;\Phi \vdash_{\mathbb{C}}^{\kappa} \mathtt{list}\,[n]^{\alpha}\ \tau \sqsubseteq \mathtt{list}\,[n']^{\alpha'}\ \tau' \hookrightarrow \underline{\lambda} e.\mathtt{map}\ \lambda z.\ \mathtt{case}(z, x.\mathtt{inl}\ \underline{\mathbf{conv}}(\underline{\mathbf{g}}(\underline{\mathbf{conv}}(x,\ \tau^{\downarrow\Box},\ \tau)),\ \tau',\ \tau'^{\downarrow\Box}), y.\mathtt{inr}\ \underline{\mathbf{g}}\ y)\ e} \ \mathbf{l2\text{-}}\mathbb{C}$$

$$\frac{\Delta;\Phi \models \mu \leq \mu' \qquad \Delta;\Phi \models \kappa \mathrel{\dot{=}} \mathtt{cost}_{\mathtt{conv}}((A)^{\mu},(A)^{\mu'})}{\Delta;\Phi \vdash_{\epsilon}^{\kappa} (A)^{\mu} \sqsubseteq (A)^{\mu'} \hookrightarrow \underline{\lambda} e.\underline{\mathbf{conv}}(e,\ (A)^{\mu},\ (A)^{\mu'})}\ \mu \qquad \frac{\Delta;\Phi_A \vdash_{\epsilon}^{\kappa} A \sqsubseteq A' \hookrightarrow \underline{\mathbf{g}} \qquad \mu = \mathbb{S} \vee \mu = \Box}{\Delta;\Phi \vdash_{\epsilon}^{\kappa} (A)^{\mu} \sqsubseteq (A')^{\mu} \hookrightarrow \underline{\mathbf{g}}} \ \mathbf{C1}$$

$$\frac{\Delta;\Phi_A \vdash_{\mathbb{S}}^{\kappa'} A \sqsubseteq A' \hookrightarrow \underline{\mathbf{g}} \qquad \Delta;\Phi \models \kappa \mathrel{\dot{=}} \kappa' \mathbin{\hat{+}} 2}{\Delta;\Phi \vdash_{\mathbb{S}}^{\kappa} (A)^{\mathbb{C}} \sqsubseteq (A')^{\mathbb{C}} \hookrightarrow \underline{\lambda} e.\mathtt{let}\ l\ =\ \mathtt{ref}\ \underline{\mathbf{g}}(!e)\ \mathtt{in}\ \mathtt{let}\ ()\ =\ \mathtt{push}([l],\ \lambda().\, \mathtt{ref}\ \underline{\mathbf{g}}(!e))\ \mathtt{in}\ l} \ \mathbf{C2\text{-}}\mathbb{S}$$

$$\frac{\Delta;\Phi \vdash_{\epsilon}^{\kappa_1} \tau_1 \sqsubseteq \tau_2 \hookrightarrow \underline{\mathbf{g_1}} \qquad \Delta;\Phi \vdash_{\epsilon}^{\kappa_2} \tau_2 \sqsubseteq \tau_3 \hookrightarrow \underline{\mathbf{g_2}} \qquad \Delta;\Phi \models \kappa \mathrel{\dot{=}} \kappa_1 \mathbin{\hat{+}} \kappa_2}{\Delta;\Phi \vdash_{\epsilon}^{\kappa} \tau_1 \sqsubseteq \tau_3 \hookrightarrow \underline{\lambda} e.\underline{\mathbf{g_2}}(\underline{\mathbf{g_1}}(e))}\mathbf{tran}$$

Figure 11: Coercion meta-function

$$\frac{}{n \approx_{\sigma}^{\mathtt{int}} n} \qquad \frac{v_1 \approx_{\sigma}^{A} v_2 \qquad \text{if } \mu = \mathbb{S} \text{ or } \mu = \Box}{v_1 \approx_{\sigma}^{(A)^{\mu}} v_2} \qquad \frac{v_s \approx_{\sigma}^{A} \sigma(l_t)}{v_s \approx_{\sigma}^{(A)^{\mathbb{C}}} l_t} \qquad \frac{}{() \approx_{\sigma}^{\mathtt{unit}} ()} \qquad \frac{v_{1l} \approx_{\sigma}^{\tau_1} v_{2l} \qquad v_{1r} \approx_{\sigma}^{\tau_2} v_{2r}}{(v_{1l}, v_{1r}) \approx_{\sigma}^{\tau_1 \times \tau_2} (v_{2l}, v_{2r})}$$

$$\frac{\models n \mathrel{\dot{=}} 0 \mathbin{\dot{\wedge}} \alpha \mathrel{\dot{=}} 0}{[] \approx_{\sigma}^{\mathtt{list}[n]^{\alpha}\ \tau} []} \qquad \frac{v_s \approx_{\sigma}^{\tau^{\downarrow\Box}} v_t \qquad vs_s \approx_{\sigma}^{\mathtt{list}[n-1]^{\alpha}\ \tau} vs_t \qquad \models 0 \mathrel{\dot{<}} n}{v_s :: vs_s \approx_{\sigma}^{\mathtt{list}[n]^{\alpha}\ \tau} \mathtt{inl}\ v_t :: vt_s}$$

$$\frac{v_s \approx_{\sigma}^{\tau} v_t \qquad vs_s \approx_{\sigma}^{\mathtt{list}[n-1]^{\alpha-1}\ \tau} vs_t \qquad \models 0 \mathrel{\dot{<}} n \qquad \models 0 \mathrel{\dot{<}} \alpha}{v_s :: vs_s \approx_{\sigma}^{\mathtt{list}[n]^{\alpha}\ \tau} \mathtt{inr}\ v_t :: vt_s} \qquad \frac{\exists I\ S. \vdash I :: S\ \wedge\ v_s \approx_{\sigma}^{\tau[I/t]} v_t}{v_s \approx_{\sigma}^{\exists t.\ \tau} v_t}$$

$$\frac{v_s \approx_{\sigma}^{\tau_1} v_t}{\mathtt{inl}\ v_s \approx_{\sigma}^{\tau_1 + \tau_2} \mathtt{inl}\ v_t} \qquad \frac{}{\mathtt{fix}\ f(x).\, e_s \approx_{\sigma}^{\tau_1 \xrightarrow{\mathbb{S}(\kappa)} \tau_2} \mathtt{fix}\ f(x).\, e_t}$$

$$\frac{\forall \sigma' \sqsupseteq \sigma\ v_c\ v_t.\ v_s \approx_{\sigma'}^{\tau_1} v_t \Rightarrow e_s[x \mapsto v_c, f \mapsto \mathtt{fix}\ f(x).\, e_s] \sim_{\sigma'}^{\tau_2} e_t[x \mapsto v_t, f \mapsto \mathtt{fix}\ f(x).\, e_t]}{\mathtt{fix}\ f(x).\, e_s \approx_{\sigma}^{\tau_1 \xrightarrow{\mathbb{C}(\kappa)} \tau_2} \mathtt{fix}\ f(x).\, e_t} \qquad \frac{}{\Lambda.\, e_s \approx_{\sigma}^{\forall t \overset{\mathbb{S}(\kappa)}{::} S.\ \tau} \Lambda.\, e_t}$$

$$\frac{\forall I\ S.\ \vdash I :: S \Rightarrow e_s \sim_{\sigma}^{\tau[I/t]} e_t}{\Lambda.\, e_s \approx_{\sigma}^{\forall t \overset{\mathbb{C}(\kappa)}{::} S.\ \tau} \Lambda.\, e_t} \qquad \frac{e_s \Downarrow v_s,\ j \qquad \forall \sigma' \sqsupseteq \sigma.\ e_t,\ \sigma' \Downarrow v_t,\ \sigma',\ \varnothing,\ c\ \wedge\ v_s \approx_{\sigma'}^{\tau} v_t}{e_s \sim_{\sigma}^{\tau} e_t}$$

Figure 12: Similarity ralation

$$\frac{}{(n, n) \approx^{\texttt{int}}_{(\sigma,\ \sigma',\ \beta)} n} \qquad \frac{(v_i, v_c) \approx^{A}_{(\sigma,\ \sigma',\ \beta)} v_t \qquad \text{if } \mu = \mathbb{S} \text{ or } \mu = \square}{(v_i, v_c) \approx^{(A)^{\mu}}_{(\sigma,\ \sigma',\ \beta)} v_t} \qquad \frac{v_i \approx^{A}_{\sigma} \sigma(l_t) \qquad v_c \approx^{A}_{\sigma+\sigma'} \sigma' + \sigma(\beta(l_t))}{(v_i, v_c) \approx^{(A)^{\mathbb{C}}}_{(\sigma,\ \sigma',\ \beta)} l_t}$$

$$\frac{}{((), ()) \approx^{\texttt{unit}}_{(\sigma,\ \sigma',\ \beta)} ()} \qquad \frac{(v_{il}, v_{cl}) \approx^{\tau_1}_{(\sigma,\ \sigma',\ \beta)} v_{tl} \qquad (v_{ir}, v_{cr}) \approx^{\tau_2}_{(\sigma,\ \sigma',\ \beta)} v_{tr}}{((v_{il}, v_{ir}), (v_{cl}, v_{cr})) \approx^{\tau_1 \times \tau_2}_{(\sigma,\ \sigma',\ \beta)} (v_{tl}, v_{tr})} \qquad \frac{\models n \doteq 0 \mathrel{\dot\wedge} \alpha \doteq 0}{([], []) \approx^{\texttt{list}[n]^{\alpha}\ \tau}_{(\sigma,\ \sigma',\ \beta)} []}$$

$$\frac{(v_i, v_c) \approx^{\tau^{\downarrow\square}}_{(\sigma,\ \sigma',\ \beta)} v_t \qquad (vs_i, vs_c) \approx^{\texttt{list}[n-1]^{\alpha}\ \tau}_{(\sigma,\ \sigma',\ \beta)} vs_t \qquad \models 0 \mathrel{\dot<} n}{(v_i :: vs_i, v_t :: vs_t) \approx^{\texttt{list}[n]^{\alpha}\ \tau}_{(\sigma,\ \sigma',\ \beta)} \texttt{inl}\ v_t :: vs_t}$$

$$\frac{(v_i, v_c) \approx^{\tau}_{(\sigma,\ \sigma',\ \beta)} v_t \qquad (vs_i, vs_c) \approx^{\texttt{list}[n-1]^{\alpha-1}\ \tau}_{(\sigma,\ \sigma',\ \beta)} vs_t \qquad \models 0 \mathrel{\dot<} n \qquad \models 0 \mathrel{\dot<} \alpha}{(v_t :: vs_t, v_c :: vs_c) \approx^{\texttt{list}[n]^{\alpha}\ \tau}_{(\sigma,\ \sigma',\ \beta)} \texttt{inr}\ v_t :: vs_t}$$

$$\frac{\exists I\ S. \vdash I :: S\ \wedge\ (v_i, v_c) \approx^{\tau[I/t]}_{(\sigma,\ \sigma',\ \beta)} v_t}{(\texttt{pack}\ v_i, \texttt{pack}\ v_c) \approx^{\exists t.\ \tau}_{(\sigma,\ \sigma',\ \beta)} v_t} \qquad \frac{(v_i, v_c) \approx^{\tau_1}_{(\sigma,\ \sigma',\ \beta)} v_t}{(\texttt{inl}\ v_i, \texttt{inl}\ v_c) \approx^{\tau_1+\tau_2}_{(\sigma,\ \sigma',\ \beta)} \texttt{inl}\ v_t}$$

$$\frac{\begin{array}{c}\forall \sigma'_i \sqsupseteq \sigma_i\ \sigma'_c \sqsupseteq \sigma_c\ \beta' > \beta\ v_i\ v_c\ v_t.\ \texttt{dom}(\beta') \setminus \texttt{dom}(\beta) \subseteq \texttt{dom}(\sigma'_i) \setminus \texttt{dom}(\sigma_i)\ \wedge\ \texttt{dom}'(\beta') \setminus \texttt{dom}'(\beta) \subseteq \texttt{dom}'(\sigma'_c) \setminus \texttt{dom}'(\sigma_c)\ \Rightarrow \\ (v_i, v_c) \approx^{\tau_1}_{(\sigma'_i,\ \sigma'_c,\ \beta)} v_t \Rightarrow \\ (e_i[x \mapsto v_i, f \mapsto \texttt{fix}\ f(x).\, e_i], e_c[x \mapsto v_c, f \mapsto \texttt{fix}\ f(x).\, e_c]) \sim^{\tau_2}_{(\sigma'_i,\ \sigma'_c,\ \beta)} e_t[x \mapsto v_t, f \mapsto \texttt{fix}\ f(x).\, e_t]\end{array}}{(\texttt{fix}\ f(x).\, e_i, \texttt{fix}\ f(x).\, e_c) \approx^{\tau_1 \xrightarrow{\mathbb{S}(\kappa)} \tau_2}_{(\sigma_i,\ \sigma_c,\ \beta)} \texttt{fix}\ f(x).\, e_t}$$

$$\frac{\texttt{fix}\ f(x).\, e_i \approx^{\tau_1 \xrightarrow{\mathbb{C}(\kappa)} \tau_2}_{\sigma} \texttt{fix}\ f(x).\, e_t \qquad \texttt{fix}\ f(x).\, e_c \approx^{\tau_1 \xrightarrow{\mathbb{C}(\kappa)} \tau_2}_{\sigma+\sigma'} \texttt{fix}\ f(x).\, \beta(e_t)}{(\texttt{fix}\ f(x).\, e_i, \texttt{fix}\ f(x).\, e_c) \approx^{\tau_1 \xrightarrow{\mathbb{C}(\kappa)} \tau_2}_{(\sigma,\ \sigma',\ \beta)} \texttt{fix}\ f(x).\, e_t}$$

$$\frac{\forall I\ S.\ \vdash I :: S \Rightarrow (e_i, e_c) \sim^{\tau[I/t]}_{(\sigma,\ \sigma',\ \beta)} e_t}{(\Lambda.\, e_i, \Lambda.\, e_c) \approx^{\forall t \overset{\mathbb{S}(\kappa)}{::} S.\ \tau}_{(\sigma,\ \sigma',\ \beta)} \Lambda.\, e_t} \qquad \frac{\Lambda.\, e_i \approx^{\forall t \overset{\mathbb{C}(\kappa)}{::} S.\ \tau}_{\sigma} \Lambda.\, e_t \qquad \Lambda.\, e_c \approx^{\forall t \overset{\mathbb{C}(\kappa)}{::} S.\ \tau}_{\sigma'+\sigma} \Lambda.\, \beta(e_t)}{(\Lambda.\, e_i, \Lambda.\, e_c) \approx^{\forall t \overset{\mathbb{C}(\kappa)}{::} S.\ \tau}_{(\sigma,\ \sigma',\ \beta)} \Lambda.\, e_t}$$

$$\frac{\begin{array}{c}\forall \sigma'_i \sqsupseteq \sigma_i\ \sigma'_c \sqsupseteq \sigma_c\ \beta' > \beta\ v_i\ v_c\ v_t.\ \texttt{dom}(\beta') \setminus \texttt{dom}(\beta) \subseteq \texttt{dom}(\sigma'_i) \setminus \texttt{dom}(\sigma_i)\ \wedge\ \texttt{dom}'(\beta') \setminus \texttt{dom}'(\beta) \subseteq \texttt{dom}'(\sigma'_c) \setminus \texttt{dom}'(\sigma_c)\ \Rightarrow \\ e_i \Downarrow v_i,\ j \Rightarrow \\ \exists\ v_s\ j'\ v_t\ \sigma_f\ D\ c\ \sigma'_f\ \beta'\ c'.\ e_s \Downarrow v_s,\ j'\ \wedge\ e_t,\ \sigma'_i \Downarrow v_t,\ \sigma_f,\ D,\ c\ \wedge\ D,\ \sigma'_c,\ \sigma_f,\ \beta \rightsquigarrow \sigma'_f,\ \beta',\ c'\ \wedge\ (v_i, v_c) \approx^{\tau}_{(\sigma_f,\ \sigma'_f,\ \beta')} v_t\end{array}}{(e_i, e_c) \sim^{\tau}_{(\sigma_i,\ \sigma_c,\ \beta)} e_t}$$

Figure 13: Two-way-similarity relation

The theorem says that given a source expression can be typed in from-scratch mode in our type system and two substitutions for its free variable that map it to similar values, then its evaluation under the first substitution and the evaluation of its translation under a store and the second substitution terminate (statements 1 and 2) and furthermore the evaluation in the target incurs cost that is not greater than the cost derived by the type system (statement 3). Moreover, the resulting values are similar under the store that results from the evaluation of the translated expression (statement 4).

The second theorem states the soundness of the translation and the cost derived by the typing derivation in change propagation mode. Similarly to the previous theorem we will consider expressions with only one free variable.

Theorem 3 (Type and translation soundness, $\mathbb{S}$ mode)
Assume that $\cdot; \cdot; x : \tau' \vdash^{\kappa}_{\mathbb{S}} e : \tau \hookrightarrow \ulcorner e \urcorner$, $(v_i,\ v_c) \approx_{(\sigma_i,\ \sigma_c,\ \beta)} v_t$, $\texttt{dom}(\beta) \subseteq \texttt{dom}(\sigma_i)$ and $\texttt{dom}'(\beta) \subseteq \texttt{dom}(\sigma_c)$. Then if $[x \mapsto v_i]e \Downarrow v'_i,\ j$ there exists $v'_c,\ v'_t,\ \sigma_f,\ \sigma'_f,\ Q,\ \beta'\ j$ and c, such that (1) $[x \mapsto v_c]e \Downarrow v'_c,\ j'$ (2) $[x \mapsto v_t]\ulcorner e \urcorner,\ \sigma_i \Downarrow v'_t,\ \sigma_f,\ Q,\ c$ (3) $Q,\ \sigma_c,\ \sigma_f,\ \beta \rightsquigarrow \sigma'_f,\ \beta',\ c'$ (4) $\models c' \mathrel{\dot\leq} \kappa$ and (5) $(v'_i,\ v'_c) \approx_{(\sigma_f,\ \sigma'_f,\ \beta')} v'_t$.

In our second theorem we assume that a source expression can be typed in change propagation mode and, furthermore, that we are given two substitutions for the source, one that represents the initial values of the inputs and the other one the updated, and one substitution for the free variable in the target expression along with two stores and a bijection between their domains, such that the values that the three substitutions are assigning to the free variable are two-way-similar under the given stores and the bijection between them. The theorem states that if the evaluation of the source expression under the first substitution terminates yielding an initial value, then the evaluation of the expression under the second substitution terminates yielding a potentially updated value (statement 1), and the evaluation the translated expression in the target terminates yielding a target value, a final store and a queue holding deferred computations (statement 2). Furthermore, propagating changes from the updated store using the queue that resulted from the evaluation yields an updated final store and a final bijection (statement 3) and moreover its cost is no more than the cost derived by the type system (statement 4). The initial and the updated values obtained by evaluation in the source language are two-way-similar with the value obtained in the target language under the two final stores and the bijection (statement 5).

To prove the two theorems above we build two relational Kripke models, one for each mode of the type system. Our relational models capture enough invariants to allow us to prove the above theorems. The models are step indexed in order to handle recursive functions.

In the expression relation of our first model we interpret a source type as pairs of source expressions and target expressions indexed by a possible word, which in this case is a store and a step index. Intuitively, this interpretation relates a source expression and a target expression that evaluate to similar vales and furthermore the evaluation of the target expression incurs cost that is less or equal to the cost represented by the type index. The relational model can be seen in fig. 14. We state and prove the fundamental property of our relational interpretation.

Theorem 4 (Fundamental property, $\mathbb{C}$ mode)
Assume that $\Delta; \Phi; \Gamma \vdash_{\mathbb{C}}^{\kappa} e : \tau \hookrightarrow \ulcorner e \urcorner$, $\varphi \in \mathcal{D}[\![\Delta]\!]$, $(\theta_s,\ \theta_t,\ (\sigma,\ m)) \in \mathcal{G}(\!|\varphi\Gamma|\!)$ and $\models \varphi\Phi$. Then $(\theta_s e,\ \theta_t \ulcorner e \urcorner,\ (\sigma,\ m)) \in \mathcal{E}(\!|\varphi\tau|\!)^{\varphi\kappa}$.

In the expression relation of our second model we interpret a source type as triples of two source expressions and a target expression indexed by a possible word, which in this case is two stores, a partial bijection between their domains, and a step index. Intuitively, in this interpretation two source expressions and a target expression are related for a given world if whenever the first source expression evaluates in steps less than the given step index then the three expressions are two-way-similar under every possible future world and furthermore change propagation does not incur a cost greater than the type index. The second relational model can be seen in fig. 15. theorem 3 and theorem 4 are corollaries of the fundamental properties.

Theorem 5 (Fundamental property, $\mathbb{S}$ mode)
Assume that $\Delta; \Phi; \Gamma \vdash_{\mathbb{C}}^{\kappa} e : \tau \hookrightarrow \ulcorner e \urcorner$, $\varphi \in \mathcal{D}[\![\Delta]\!]$, $(\theta_i,\ \theta_c,\ \theta_t,\ (\sigma_i,\ \sigma_c,\ \beta,\ m)) \in \mathcal{G}[\![\varphi\Gamma]\!] \models \varphi\Phi$, $\mathtt{dom}(\beta) \subseteq \mathtt{dom}(\sigma_i)$ and $\mathtt{dom}'(\beta) \subseteq \mathtt{dom}(\sigma_c)$. Then $(\theta_i e,\ \theta_c e,\ \theta_t \ulcorner e \urcorner,\ (\sigma_i,\ \sigma_c,\ \beta,\ m)) \in \mathcal{E}[\![\varphi\tau]\!]^{\varphi\kappa}$

Note that both of our models enjoy monotonicity, meaning that if a pair or a triple or expressions are related in a world, then they are related in all possible future worlds, i.e. all the possible extensions of the current world. This property is crucial in the proof of the fundamental property.

5 Related Work

Çiçek *et al.* [5], in their seminal work on CostIt, provided a soundness proof for a preliminary version of the type system. The soundness proof of the type system in [5] was with respect to an abstract cost model of change propagation without justification of its realizability. In this work we provide a soundness proof that shows that the cost derived by the type system is an upper bound of the actual cost that is incurred by our change propagation mechanism.

Our translation methodology resembles the methodology followed by Chen *et al.* [4] in their work on implicit self-adjusting computation. In this work, a functional program with changeability annotations is

$$
\begin{aligned}
&\text{World} = \text{Store} \times \text{Step index}\\
&\mathcal{V}\llparenthesis\tau\rrparenthesis, \mathcal{V}_A\llparenthesis\tau\rrparenthesis \subseteq \text{Source Value} \times \text{Target Value} \times \text{World}\\
&\mathcal{E}\llparenthesis\tau\rrparenthesis^{\kappa} \subseteq \text{Source Expression} \times \text{Target Expression} \times \text{World}
\end{aligned}
$$

$$
\begin{aligned}
&\sigma \sqsupseteq \sigma' = \forall\ l \in \mathtt{dom}(\sigma').\ l \in \mathtt{dom}(\sigma)\ \wedge\ \sigma(l) = \sigma'(l)\\
&(\sigma,\ j) > (\sigma',\ j') = \sigma \sqsupseteq \sigma'\ \wedge\ j < j'
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{V}\llparenthesis(A)^{\mu}\rrparenthesis &= \mathcal{V}_A\llparenthesis A\rrparenthesis \qquad \text{if } \mu = \mathbb{S} \text{ or } \mu = \square\\
\mathcal{V}\llparenthesis(A)^{\mathbb{C}}\rrparenthesis &= \{\ (v_s,\ l,\ (\sigma,\ k)) \mid (v_i,\ \sigma(l),\ (\sigma,\ k)) \in \mathcal{V}_A\llparenthesis A\rrparenthesis\ \}\\
\mathcal{V}\llparenthesis\mathtt{unit}\rrparenthesis &= \{\ ((),\ (),\ W) \mid \top\ \}\\
\mathcal{V}\llparenthesis\tau_1 \times \tau_2\rrparenthesis &= \{\ ((v_{s1}, v_{s2}),\ (v_{t1}, v_{t2}),\ W) \mid (v_{s1},\ v_{t1},\ W) \in \mathcal{V}\llparenthesis\tau_1\rrparenthesis\ \wedge\ (v_{s2},\ v_{t2},\ W) \in \mathcal{V}\llparenthesis\tau_2\rrparenthesis\ \}\\
\mathcal{V}\llparenthesis\mathtt{list}\,[n]^{\alpha}\ \tau\rrparenthesis &= \{\ ([],\ [],\ \sigma)k \mid\ \models n \doteq 0\ \dot{\wedge}\ \alpha \doteq 0\ \}\\
\mathcal{V}\llparenthesis\mathtt{list}\,[n]^{\alpha}\ \tau\rrparenthesis &= \{\ (v_s :: vs_s,\ \mathtt{inl}\ v_t :: vs_t,\ W) \mid (v_s,\ v_t,\ W) \in \mathcal{V}\llparenthesis\tau^{\downarrow\square}\rrparenthesis\ \wedge\\
&\qquad\qquad (vs_s,\ vs_t,\ W) \in \mathcal{V}\llparenthesis\mathtt{list}\,[n-1]^{\alpha}\ \tau\rrparenthesis\ \wedge\ \models 0 \dot{<} n\ \}\cup\\
&\quad\ \{\ (v_s :: vs_s,\ \mathtt{inr}\ v_t :: vs_t,\ W) \mid (v_s,\ v_t,\ W) \in \mathcal{V}\llparenthesis\tau\rrparenthesis\ \wedge\\
&\qquad\qquad (vs_s,\ vs_t,\ W) \in \mathcal{V}\llparenthesis\mathtt{list}\,[n-1]^{\alpha-1}\ \tau\rrparenthesis\ \wedge\ \models 0 \dot{<} n\ \dot{\wedge}\ 0 \dot{<} \alpha\ \}\\
\mathcal{V}\llparenthesis\exists t.\ \tau\rrparenthesis &= \{\ (\mathtt{pack}\ v_s,\ v_t,\ W) \mid \exists I.\ \vdash I :: S\ \wedge\ (v_s,\ v_t,\ W) \in \mathcal{V}\llparenthesis\tau[I/t]\rrparenthesis\ \}
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{V}_A\llparenthesis\mathtt{int}\rrparenthesis &= \{\ (n,\ n,\ W) \mid \top\ \}\\
\mathcal{V}_A\llparenthesis\tau_1 + \tau_2\rrparenthesis &= \{\ (\mathtt{inl}\ v_s,\ \mathtt{inl}\ v_t,\ W) \mid (v_s,\ v_t,\ W) \in \mathcal{V}\llparenthesis\tau_1\rrparenthesis\ \}\cup\\
&\quad\ \{\ (\mathtt{inr}\ v_s,\ \mathtt{inr}\ v_t,\ W) \mid (v_s,\ v_t,\ W) \in \mathcal{V}\llparenthesis\tau_2\rrparenthesis\ \}\\
\mathcal{V}_A\llparenthesis\tau_1 \xrightarrow{\mathbb{S}(\kappa)} \tau_2\rrparenthesis &= \{\ (\mathtt{fix}\ f(x).\, e_s,\ \mathtt{fix}\ f(x).\, e_t,\ W) \mid \top\ \}\\
\mathcal{V}_A\llparenthesis\tau_1 \xrightarrow{\mathbb{C}(\kappa)} \tau_2\rrparenthesis &= \{\ (\mathtt{fix}\ f(x).\, e_s,\ \mathtt{fix}\ f(x).\, e_t,\ W) \mid\\
&\qquad \forall W' > W.\ v_t, (v_s,\ v_t,\ W') \in \mathcal{V}\llparenthesis\tau_1\rrparenthesis \Rightarrow\\
&\qquad\quad ([x \mapsto v_s, f \mapsto \mathtt{fix}\ f(x).\, e_s]e_s,\ [x \mapsto v_t, f \mapsto \mathtt{fix}\ f(x).\, e_t]e_t,\ W') \in \mathcal{E}\llparenthesis\tau_2\rrparenthesis^{\kappa}\ \}\\
\mathcal{V}_A\llparenthesis\forall t \overset{\mathbb{S}(\kappa)}{::} S.\ \tau\rrparenthesis &= \{\ (\Lambda.\, e_s,\ \Lambda.\, e_t,\ W) \mid \top\ \}\\
\mathcal{V}_A\llparenthesis\forall t \overset{\mathbb{C}(\kappa)}{::} S.\ \tau\rrparenthesis &= \{\ (\Lambda.\, e_s,\ \Lambda.\, e_t,\ W) \mid \forall I.\ \vdash I :: S \Rightarrow (e_s,\ e_t,\ W) \in \mathcal{E}\llparenthesis\tau[I/t]\rrparenthesis^{\kappa[I/t]}\ \}
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{E}\llparenthesis\tau\rrparenthesis^{\kappa} &= \{\ (e_s,\ e_t,\ (\sigma,\ k)) \mid \exists v_s\ j.\ e_s \Downarrow v_s,\ j\ \wedge\\
&\qquad j < k \Rightarrow \forall\sigma' \sqsupseteq \sigma.\exists\ v_t\ \sigma''\ c\ .e_t,\ \sigma'' \Downarrow v_t,\ \sigma'',\ \varnothing,\ c\ \wedge\ \models c\ \dot{\leq}\ \kappa\ \wedge\ (v_s,\ v_t,\ (\sigma'',\ k-j)) \in \mathcal{V}\llparenthesis\tau\rrparenthesis\ \}
\end{aligned}
$$

Figure 14: Unary step-indexed interpretation of types

translated to a self-adjusting program. They prove the functional correctness of the derived program and also that preserves the intentional semantics of the original program, i.e. that change propagation will not perform worse that the evaluation of the original program. However, the novelty of CostIt lies on the ability to derive upper bounds on the cost of incremental computation complexity, that is often asymptotically faster that from-scratch evaluation. The upper bound derived by CostIt is often much more precise than the execution time of the original program and, consequently, our soundness proof provides a tighter upper bound on the execution time of the translated program.

Ley-Wild *et al.* [11] provide a way to reason about the effectiveness of change propagation time by comparing the edit distance of program traces. They provide a translation scheme from source programs to self-adjusting target programs. They prove that their translation preserves the semantics of the original program and also that change propagation between two runs has a cost that is upper bounded by the trace distance of the two execution traces. The main difference of this work compared to CostIt is that it requires direct analysis on the cost semantics of programs. In contrast, CostIt provides language support for reasoning statically about the cost of incremental computation.

6 Improvements and Future Work

Besides the longer term future directions that we mentioned in the beginning, our next step aims in improving the precision of our cost analysis. More precisely, we plan to modify the cost derived by the type system in

World = Store × Store × Partial Bijection
$\mathcal{V}\llbracket\tau\rrbracket, \mathcal{V}_A\llbracket\tau\rrbracket \subseteq$ Source Value × Source Value × Target Value × World × Step index
$\mathcal{E}\llbracket\tau\rrbracket^{\kappa} \subseteq$ Source Expression × Source Expression × Target Expression × World × Step index

$$(\sigma_i,\ \sigma_c,\ \beta,\ j) > (\sigma'_i,\ \sigma'_c,\ \beta',\ j') = \sigma_i \sqsupseteq \sigma'_i \wedge \sigma_c \sqsupseteq \sigma'_c \wedge \mathtt{dom}(\beta)\backslash\mathtt{dom}(\beta') \subseteq \mathtt{dom}(\sigma_i)\backslash\mathtt{dom}(\sigma'_i) \wedge \mathtt{dom}'(\beta)\backslash\mathtt{dom}'(\beta') \subseteq \mathtt{dom}(\sigma_c)\backslash\mathtt{dom}(\sigma'_c) \wedge j < j'$$

$$\begin{aligned}
\mathcal{V}\llbracket(A)^{\mu}\rrbracket &= \mathcal{V}_A\llbracket A\rrbracket \qquad \text{if } \mu = \mathbb{S} \text{ or } \mu = \square \\
\mathcal{V}\llbracket(A)^{\mathbb{C}}\rrbracket &= \{\ (v_i,\ v_c,\ l,\ (\sigma_i,\ \sigma_c,\ \beta,\ k)) \mid (v_i,\ l,\ (\sigma_i,\ k)) \in \mathcal{V}\llparenthesis(A)^{\mathbb{C}}\rrparenthesis \wedge \forall\ k.(v_c,\ \beta(l),\ (\sigma_c + \sigma_i,\ k)) \in \mathcal{V}\llparenthesis(A)^{\mathbb{C}}\rrparenthesis\ \} \\
\mathcal{V}\llbracket\mathtt{unit}\rrbracket &= \{\ ((),\ (),\ (),\ W) \mid \top\ \} \\
\mathcal{V}\llbracket\tau_1 \times \tau_2\rrbracket &= \{\ ((v_{i1}, v_{i2}),\ (v_{c1}, v_{c2}),\ (v_{t1}, v_{t2}),\ W) \mid (v_{i1},\ v_{c1},\ v_{t1},\ W) \in \mathcal{V}\llbracket\tau_1\rrbracket\ \wedge\ (v_{i2},\ v_{c2},\ v_{t2},\ W) \in \mathcal{V}\llbracket\tau_2\rrbracket\ \} \\
\mathcal{V}\llbracket\mathtt{list}\,[n]^{\alpha}\ \tau\rrbracket &= \{\ ([],\ [],\ [],\ W) \mid\ \models n \doteq 0\ \dot{\wedge}\ \alpha \doteq 0\ \} \\
\mathcal{V}\llbracket\mathtt{list}\,[n]^{\alpha}\ \tau\rrbracket &= \{\ (v_i :: vs_i,\ v_c :: vs_c,\ \mathtt{inl}\ v_t :: vs_t,\ W) \mid \\
&\qquad\qquad (v_i,\ v_c,\ v_t,\ W) \in \mathcal{V}\llbracket\tau^{\downarrow\square}\rrbracket\ \wedge (vs_i,\ vs_c,\ vs_t,\ W) \in \mathcal{V}\llbracket\mathtt{list}\,[n-1]^{\alpha}\ \tau\rrbracket\ \wedge\ \models 0\ \dot{<}\ n\ \} \cup \\
&\quad\ \{\ (v_i :: vs_i,\ v_c :: vs_c,\ \mathtt{inr}\ v_t :: vs_t,\ W) \mid \\
&\qquad\qquad (v_i,\ v_c,\ v_t,\ W) \in \mathcal{V}\llbracket\tau\rrbracket\ \wedge (vs_i,\ vs_c,\ vs_t,\ W) \in \mathcal{V}\llbracket\mathtt{list}\,[n-1]^{\alpha-1}\ \tau\rrbracket\ \wedge\ \models 0\ \dot{<}\ n\ \dot{\wedge}\ 0\ \dot{<}\ \alpha\ \} \\
\mathcal{V}\llbracket\exists t.\ \tau\rrbracket &= \{\ (\mathtt{pack}\ v_i,\ \mathtt{pack}\ v_c,\ v_t,\ W) \mid \exists I.\ \vdash I :: S\ \wedge\ (v_i,\ v_c,\ v_t,\ W) \in \mathcal{V}\llbracket\tau[I/t]\rrbracket\ \}
\end{aligned}$$

$$\begin{aligned}
\mathcal{V}_A\llbracket\mathtt{int}\rrbracket &= \{\ (n,\ n,\ n,\ W) \mid \top\ \} \\
\mathcal{V}_A\llbracket\tau_1 + \tau_2\rrbracket &= \{\ (\mathtt{inl}\ v_i,\ \mathtt{inl}\ v_c,\ \mathtt{inl}\ v_t,\ W) \mid (v_i,\ v_c,\ v_t,\ W) \in \mathcal{V}\llbracket\tau_1\rrbracket\ \} \cup \\
&\quad\ \{\ (\mathtt{inr}\ v_i,\ \mathtt{inr}\ v_c,\ \mathtt{inr}\ v_t,\ W) \mid (v_i,\ v_c,\ v_t,\ W) \in \mathcal{V}\llbracket\tau_2\rrbracket\ \} \\
\mathcal{V}_A\llbracket\tau_1 \xrightarrow{\mathbb{S}(\kappa)} \tau_2\rrbracket &= \{\ (\mathtt{fix}\ f(x).\,e_i,\ \mathtt{fix}\ f(x).\,e_c,\ \mathtt{fix}\ f(x).\,e_t,\ W) \mid \\
&\qquad\qquad \forall\ W' \geq W.\ v_i,\ v_c,\ v_t, (v_i,\ v_c,\ v_t,\ W') \in \mathcal{V}\llbracket\tau_1\rrbracket \Rightarrow \\
&\qquad\qquad\quad ([x \mapsto v_i, f \mapsto \mathtt{fix}\ f(x).\,e_i]e_i,\ [x \mapsto v_c, f \mapsto \mathtt{fix}\ f(x).\,e_c]e_c,\ [x \mapsto v_t, f \mapsto \mathtt{fix}\ f(x).\,e_t]e_t,\ W') \in \mathcal{E}\llbracket\tau_2\rrbracket^{\kappa}\ \} \\
\mathcal{V}_A\llbracket\tau_1 \xrightarrow{\mathbb{C}(\kappa)} \tau_2\rrbracket &= \{\ (\mathtt{fix}\ f(x).\,e_i,\ \mathtt{fix}\ f(x).\,e_c,\ \mathtt{fix}\ f(x).\,e_t,\ (\sigma_i,\ \sigma_c,\ \beta,\ k)) \mid (\mathtt{fix}\ f(x).\,e_i,\ \mathtt{fix}\ f(x).\,e_t,\ (\sigma_i,\ k)) \in \mathcal{V}_A\llparenthesis\tau_1 \xrightarrow{\mathbb{C}(\kappa)} \tau_2\rrparenthesis\ \wedge \\
&\qquad\qquad \forall k.\ (\mathtt{fix}\ f(x).\,e_c,\ \mathtt{fix}\ f(x).\,\beta(e_t),\ (\sigma_c + \sigma_i,\ k)) \in \mathcal{V}_A\llparenthesis\tau_1 \xrightarrow{\mathbb{C}(\kappa)} \tau_2\rrparenthesis\ \} \\
\mathcal{V}_A\llbracket\forall t \overset{\mathbb{S}(\kappa)}{::} S.\ \tau\rrbracket &= \{\ (\Lambda.\,e_i,\ \Lambda.\,e_c,\ \Lambda.\,e_t,\ W) \mid \forall I.\ \vdash I :: S \Rightarrow (e_i,\ e_c,\ e_t,\ W) \in \mathcal{E}\llbracket\tau[I/t]\rrbracket^{\kappa[I/t]}\ \} \\
\mathcal{V}_A\llbracket\forall t \overset{\mathbb{C}(\kappa)}{::} S.\ \tau\rrbracket &= \{\ (\Lambda.\,e_i,\ \Lambda.\,e_c,\ \Lambda.\,e_t,\ (\sigma_i,\ \sigma_c,\ \beta,\ k)) \mid (\Lambda.\,e_i,\ \Lambda.\,e_t,\ (\sigma_i,\ k)) \in \mathcal{V}_A\llparenthesis\forall t \overset{\mathbb{C}(\kappa)}{::} S.\ \tau\rrparenthesis\ \wedge \\
&\qquad\qquad \forall k.\ (\Lambda.\,e_c,\ \Lambda.\,\beta(e_t),\ (\sigma_c + \sigma_i,\ k)) \in \mathcal{V}_A\llparenthesis\tau_1 \xrightarrow{\mathbb{C}(\kappa)} \tau_2\rrparenthesis\ \}
\end{aligned}$$

$$\begin{aligned}
\mathcal{E}\llparenthesis\tau\rrparenthesis^{\kappa} &= \{\ (e_i,\ e_c,\ e_t,\ W) \mid \forall j_i\ \sigma_i\ \sigma_c\ \beta\ v_i.\ (\sigma_i,\ \sigma_c,\ \beta,\ j_i) \geq W\ \wedge\ e_i \Downarrow v_i,\ j_i \Rightarrow \\
&\qquad\qquad \exists v_c\ j_c\ v_t\ \sigma_f\ D\ c\ \sigma'_f\ \beta_f\ c'.\ e_c \Downarrow v_c,\ j_c\ \wedge \\
&\qquad\qquad\quad e_t,\ \sigma_i \Downarrow\ v_t,\ \sigma_f,\ Q,\ j_c \wedge\ Q,\ \sigma_c,\ \sigma_f,\ \beta \rightsquigarrow \sigma'_f,\ \beta',\ c'\ \wedge \\
&\qquad\qquad\quad (\sigma_f,\ \sigma'_f,\ \beta') \geq (\sigma_i,\ \sigma_c,\ \beta)\ \wedge\ \models c\ \dot{\leq}\ \kappa\ \wedge\ (v_i,\ v_c,\ v_t,\ (\sigma_f,\ \sigma'_f,\ \beta',\ k - j_i)) \in \mathcal{V}\llbracket\tau\rrbracket\ \}
\end{aligned}$$

Figure 15: Step-indexed interpretation of types

order to account for the cost incurred by queue operations and locations update during change propagation. Currently, the type system tracks only the cost of pure computation during change propagation, i.e. the cost of evaluating the recorded closures. We anticipate that there is no serious difficultly in this direction.

Another possible direction, which we did not consider during this internship due to time constraints, would be the mechanization of CostIt's metatheory in a proof assistant. We believe that, with sufficient automation, the cumbersomeness of our handwritten proofs could be significantly reduced.

References

[1] U. A. Acar, G. E. Blelloch, M. Blume, R. Harper, and K. Tangwongsan. An experimental analysis of self-adjusting computation. *ACM Trans. Program. Lang. Syst.*, 32(1):3:1–3:53, 2009.

[2] U. A. Acar, G. E. Blelloch, and R. Harper. Adaptive functional programming. volume 28, pages 990–1034. ACM, 2006.

[3] A. J. Ahmed. *Semantics of Types for Mutable State.* PhD thesis, Princeton University, 2004.

[4] Y. Chen, J. Dunfield, M. A. Hammer, and U. A. Acar. Implicit self-adjusting computation for purely functional programs. In *Int'l Conference on Functional Programming (ICFP '11)*, pages 129–141, Sept. 2011.

[5] E. Çiçek, D. Garg, and U. A. Acar. Refinement types for incremental computational complexity. In *Programming Languages and Systems - 24th European Symposium on Programming, ESOP 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*, pages 406–431, 2015.

[6] A. Demers, T. Reps, and T. Teitelbaum. Incremental evaluation for attribute grammars with application to syntax-directed editors. In *Proceedings of the 8th Symposium on Principles of Programming Languages*, POPL '81, pages 105–116. ACM, 1981.

[7] D. Dreyer, A. Ahmed, and L. Birkedal. Logical step-indexed logical relations. *Logical Methods in Computer Science*, 7(2), 2011.

[8] M. Gaboardi, A. Haeberlen, J. Hsu, A. Narayan, and B. C. Pierce. Linear dependent types for differential privacy. In *Proceedings of the 40th Annual Symposium on Principles of Programming Languages*, POPL '13, pages 357–370. ACM, 2013.

[9] M. A. Hammer, K. Y. Phang, M. Hicks, and J. S. Foster. Adapton: Composable, demand-driven incremental computation. In *Proceedings of the 35th Conference on Programming Language Design and Implementation*, PLDI '14, pages 156–166. ACM, 2014.

[10] A. Heydon, R. Levin, and Y. Yu. Caching function calls using precise dependencies. In *Proceedings of the Conference on Programming Language Design and Implementation*, PLDI '00, pages 311–320. ACM, 2000.

[11] R. Ley-Wild, U. A. Acar, and M. Fluet. A cost semantics for self-adjusting computation. In *Proceedings of the 36th Annual Symposium on Principles of Programming Languages*, POPL '09, pages 186–199. ACM, 2009.

[12] A. Nanevski and F. Pfenning. Staged computation with names and necessity. *J. Funct. Program.*, 15(6):893–939, 2005.

[13] F. Nielson and H. Nielson. Type and effect systems. In *Correct System Design*, volume 1710 of *Lecture Notes in Computer Science*, pages 114–136. Springer-Verlag, 1999.

[14] F. Pottier and V. Simonet. Information flow inference for ML. *ACM Trans. Program. Lang. Syst.*, 25(1):117–158, 2003.

[15] W. Pugh. *Incremental Computation via Function Caching.* PhD thesis, Department of Computer Science, Cornell University, Aug. 1988.

[16] J. Thamsborg and L. Birkedal. A kripke logical relation for effect-based program transformations. In *Proceedings of the 16th ACM SIGPLAN International Conference on Functional Programming*, ICFP '11, pages 445–456, New York, NY, USA, 2011. ACM.

[17] H. Xi and F. Pfenning. Dependent types in practical programming. In *Proceedings of the 26th Symposium on Principles of Programming Languages*, POPL '99, pages 214–227. ACM, 1999.

[18] D. Yellin and R. Strom. Inc: A language for incremental computations. In *Proceedings of the Conference on Programming Language Design and Implementation*, PLDI '88, pages 115–124. ACM, 1988.

Appendix A Definitions

$$
\begin{array}{lcl}
\|\texttt{int}\|^A & = & \texttt{int} \\
\|\tau_1 + \tau_2\|^A & = & \|\tau_1\| + \|\tau_2\| \\
\|\tau_1 \xrightarrow{\mu(\kappa)} \tau_2\|^A & = & \|\tau_1\| \to \|\tau_2\| \\
\|\forall i \overset{\mu(\kappa)}{::} S.\ \tau\|^A & = & \texttt{unit} \to \|\tau\| \\
\|(A)^{\mathbb{C}}\| & = & \texttt{ref}\ \|A\|^A \\
\|(A)^{\mathbb{S}}\| & = & \|A\|^A \\
\|(A)^{\square}\| & = & \|A\|^A \\
\|\texttt{unit}\| & = & \texttt{unit} \\
\|\tau_1 \times \tau_2\| & = & \|\tau_1\| \times \|\tau_2\| \\
\|\texttt{list}\,[n]^\alpha\ \tau\| & = & \texttt{list}\ \|\tau\| + \|\tau^{\downarrow\square}\| \\
\|\exists i.\ \tau\| & = & \|\tau\| \\
\|C \to \tau\| & = & \|\tau\| \\
\|C \wedge \tau\| & = & \|\tau\|
\end{array}
$$

Figure 16: Translation of types

$$
\begin{array}{lcl}
\texttt{map}\ f\ e & = & \texttt{case}_{\mathrm{L}}\ e\ \texttt{of} \\
 & & \mid []\ \to\ [] \\
 & & \mid h :: tl\ \to f\ h :: \texttt{map}\ f\ tl
\end{array}
$$

Figure 17: Mapping function in the target language

$$
\begin{array}{lcl}
\texttt{foldr}\ f\ e\ a & = & \texttt{case}_{\mathrm{L}}\ e\ \texttt{of} \\
 & & \mid []\ \to\ a \\
 & & \mid h :: tl\ \to \texttt{let}\ a' = \texttt{foldr}\ f\ tl\ a\ \texttt{in}\ f\ h\ a'
\end{array}
$$

Figure 18: Folding function in the target language

Definition 1 (Index term interpretation)
Let $\varphi \in \mathcal{D}[\![\Delta]\!]$ and $\Delta; \Phi \vdash i : S$. The index term i is interpreted as follows

$$
\begin{array}{lcl}
\underline{\mathbf{deep}_{\mathbf{aux}}}(e,\ acc,\ (A)^{\mathbb{C}}) & = & \mathtt{let}\ l\ =\ \mathtt{ref}\ !e\ \mathtt{in}\ (l, \mathtt{cons}\ l\ acc) \\
\underline{\mathbf{deep}_{\mathbf{aux}}}(e,\ acc,\ \tau_1 \times \tau_2) & = & \mathtt{let}\ v\ =\ e\ \mathtt{in} \\
 & & \mathtt{let}\ p_1\ =\ \underline{\mathbf{deep}_{\mathbf{aux}}}(\mathtt{fst}\ v,\ acc,\ \tau_1)\ \mathtt{in} \\
 & & \mathtt{let}\ acc'\ =\ \mathtt{snd}\ p_1\ \mathtt{in} \\
 & & \mathtt{let}\ p_2\ =\ \underline{\mathbf{deep}_{\mathbf{aux}}}(\mathtt{snd}\ v,\ acc',\ \tau_2)\ \mathtt{in} \\
 & & ((\mathtt{fst}\ p_1, \mathtt{fst}\ p_2), \mathtt{snd}\ p_2) \\
\underline{\mathbf{deep}_{\mathbf{aux}}}(e,\ acc,\ \mathtt{unit}) & = & acc \\
\underline{\mathbf{deep}_{\mathbf{aux}}}(e,\ acc,\ \exists i.\ \tau) & = & \underline{\mathbf{deep}_{\mathbf{aux}}}(e,\ acc,\ \tau) \\
\underline{\mathbf{deep}_{\mathbf{aux}}}(e,\ acc,\ C \to \tau) & = & \underline{\mathbf{deep}_{\mathbf{aux}}}(e,\ acc,\ \tau) \\
\underline{\mathbf{deep}_{\mathbf{aux}}}(e,\ acc,\ C \wedge \tau) & = & \underline{\mathbf{deep}_{\mathbf{aux}}}(e,\ acc,\ \tau) \\
\underline{\mathbf{deep}_{\mathbf{aux}}}(e,\ acc,\ \mathtt{list}\,[n]^{\alpha}\ \tau) & = & \mathtt{foldr} \\
 & & \quad (\lambda h.\, \lambda a.\, \mathtt{case}(h, h_l.\bot, h_r.\mathtt{let}\ p\ =\ \underline{\mathbf{deep}_{\mathbf{aux}}}(h_r,\ \tau,\ a)\ \mathtt{in}\ (\mathtt{cons}\ (\mathtt{inr}\ (\mathtt{fst}\ p))\ (\\
 & & \quad e\ ([], acc)
\end{array}
$$

$$
\underline{\mathbf{deep}}(e,\ \tau)\ \ =\ \ \underline{\mathbf{deep}_{\mathbf{aux}}}(e,\ [],\ \tau)
$$

Figure 19: Deep dereferencing, re-referencing and flattening of the locations that occur in an expression with expression reconstruction

$$
\begin{array}{lcl}
\mathtt{cost}_{\mathtt{deep}}((A)^{\mathbb{C}}) & = & 3 \\
\mathtt{cost}_{\mathtt{deep}}(\tau_1 \times \tau_2) & = & \mathtt{cost}_{\mathtt{deep}}(\tau_1)\ \hat{+}\ \mathtt{cost}_{\mathtt{deep}}(\tau_2)\ \hat{+}\ 10 \\
\mathtt{cost}_{\mathtt{deep}}(\mathtt{unit}) & = & 0 \\
\mathtt{cost}_{\mathtt{deep}}(\exists i.\ \tau) & = & \mathtt{cost}_{\mathtt{deep}}(\tau) \\
\mathtt{cost}_{\mathtt{deep}}(C \to \tau) & = & \mathtt{cost}_{\mathtt{deep}}(\tau) \\
\mathtt{cost}_{\mathtt{deep}}(C \wedge \tau) & = & \mathtt{cost}_{\mathtt{deep}}(\tau) \\
\mathtt{cost}_{\mathtt{deep}}(\mathtt{list}\,[n]^{\alpha}\ \tau) & = & n\ \hat{\cdot}\ (\mathtt{cost}_{\mathtt{deep}}(\tau)\ \hat{+}\ 10)
\end{array}
$$

Figure 20: The cost associated with $\underline{\mathbf{deep}}(\cdot,\ \cdot)$

$$
\begin{array}{lcl}
\underline{\mathbf{deep}'_{\mathbf{aux}}}(e,\ acc,\ (A)^{\mathbb{C}}) & = & \mathtt{let}\ l\ =\ \mathtt{ref}\ !e\ \mathtt{in}\ \mathtt{cons}\ l\ acc \\
\underline{\mathbf{deep}'_{\mathbf{aux}}}(e,\ acc,\ \tau_1 \times \tau_2) & = & \mathtt{let}\ v\ =\ e\ \mathtt{in} \\
 & & \mathtt{let}\ acc'\ =\ \underline{\mathbf{deep}'_{\mathbf{aux}}}(\mathtt{fst}\ v,\ acc,\ \tau_1)\ \mathtt{in} \\
 & & \underline{\mathbf{deep}'_{\mathbf{aux}}}(\mathtt{snd}\ v,\ acc',\ \tau_2) \\
\underline{\mathbf{deep}'_{\mathbf{aux}}}(e,\ acc,\ \mathtt{unit}) & = & acc \\
\underline{\mathbf{deep}'_{\mathbf{aux}}}(e,\ acc,\ \exists i.\ \tau) & = & \underline{\mathbf{deep}'_{\mathbf{aux}}}(e,\ acc,\ \tau) \\
\underline{\mathbf{deep}'_{\mathbf{aux}}}(e,\ acc,\ C \to \tau) & = & \underline{\mathbf{deep}'_{\mathbf{aux}}}(e,\ acc,\ \tau) \\
\underline{\mathbf{deep}'_{\mathbf{aux}}}(e,\ acc,\ C \wedge \tau) & = & \underline{\mathbf{deep}'_{\mathbf{aux}}}(e,\ acc,\ \tau) \\
\underline{\mathbf{deep}'_{\mathbf{aux}}}(e,\ acc,\ \mathtt{list}\,[n]^{\alpha}\ \tau) & = & \mathtt{foldr}\ (\lambda h.\, \lambda a.\, \mathtt{case}(h, h_l.\bot, h_r.\mathtt{inr}\ \underline{\mathbf{deep}'_{\mathbf{aux}}}(h_r,\ \tau,\ a)))\ e\ acc
\end{array}
$$

$$
\underline{\mathbf{deep}'}(e,\ \tau)\ \ =\ \ \underline{\mathbf{deep}'_{\mathbf{aux}}}(e,\ [],\ \tau)
$$

Figure 21: Deep dereferencing, re-referencing and flattening of the locations that occur in an expression without expression reconstruction

$$
\begin{array}{lcl}
\texttt{cost}_{\texttt{deepref}}((A)^{\mathbb{C}}) & = & 3 \\
\texttt{cost}_{\texttt{deepref}}(\tau_1 \times \tau_2) & = & \texttt{cost}_{\texttt{deep}}(\tau_1) \mathbin{\hat{+}} \texttt{cost}_{\texttt{deep}}(\tau_2) \mathbin{\hat{+}} 4 \\
\texttt{cost}_{\texttt{deepref}}(\texttt{unit}) & = & 0 \\
\texttt{cost}_{\texttt{deepref}}(\exists i.\ \tau) & = & \texttt{cost}_{\texttt{deepref}}(\tau) \\
\texttt{cost}_{\texttt{deepref}}(C \to \tau) & = & \texttt{cost}_{\texttt{deepref}}(\tau) \\
\texttt{cost}_{\texttt{deepref}}(C \wedge \tau) & = & \texttt{cost}_{\texttt{deepref}}(\tau) \\
\texttt{cost}_{\texttt{deepref}}(\texttt{list}\,[n]^{\alpha}\ \tau) & = & n \mathbin{\hat{\cdot}} (\texttt{cost}_{\texttt{deepref}}(\tau) \mathbin{\hat{+}} 6)
\end{array}
$$

Figure 22: The cost associated with $\underline{\mathbf{deep'}}(\cdot,\ \cdot)$

$$
\begin{array}{lcll}
\underline{\mathbf{conv}}(e,\ (A)^{\mu},\ (A)^{\mu}) & = & e & \\
\underline{\mathbf{conv}}(e,\ (A)^{\mu},\ (A)^{\mathbb{C}}) & = & \texttt{let}\ v\ =\ e\ \texttt{in}\ \texttt{ref}\ v & \\
\underline{\mathbf{conv}}(e,\ (A)^{\mathbb{C}},\ (A)^{\mu}) & = & !e & \text{if } (\mu = \mathbb{S}) \vee (\mu = \square) \\
\underline{\mathbf{conv}}(e,\ \tau_1 \times \tau_2,\ \tau_1' \times \tau_2') & = & (\underline{\mathbf{conv}}(\texttt{fst}\ e,\ \tau_1,\ \tau_1'), \underline{\mathbf{conv}}(\texttt{snd}\ e,\ \tau_2,\ \tau_2')) & \\
\underline{\mathbf{conv}}(e,\ \texttt{unit},\ \texttt{unit}) & = & e & \\
\underline{\mathbf{conv}}(e,\ \exists i.\ \tau,\ \exists i.\ \tau') & = & \underline{\mathbf{conv}}(e,\ \tau,\ \tau') & \\
\underline{\mathbf{conv}}(e,\ C \to \tau,\ C \to \tau') & = & \underline{\mathbf{conv}}(e,\ \tau,\ \tau') & \\
\underline{\mathbf{conv}}(e,\ C \wedge \tau,\ C \wedge \tau') & = & \underline{\mathbf{conv}}(e,\ \tau,\ \tau') & \\
\underline{\mathbf{conv}}(e,\ \texttt{list}\,[n]^{\alpha}\ \tau,\ \texttt{list}\,[n]^{\alpha}\ \tau') & = & \multicolumn{2}{l}{\texttt{map}\ (\lambda h.\, \texttt{case}(h, h_l.\underline{\mathbf{conv}}(h_l,\ \tau^{\downarrow\square},\ \tau'^{\downarrow\square}), h_r.\underline{\mathbf{conv}}(h_r,\ \tau,\ \tau')))\ e}
\end{array}
$$

Figure 23: Conversion an expression of type τ to a type τ' when the two types have the same structure same structure

$$
\begin{array}{lcl}
\texttt{cost}_{\texttt{conv}}((A)^{\mu}, (A)^{\mu}) & = & 0 \\
\texttt{cost}_{\texttt{conv}}((A)^{\mu}, (A)^{\mathbb{C}}) & = & 2 \\
\texttt{cost}_{\texttt{conv}}((A)^{\mathbb{C}}, (A)^{\mu}) & = & 1 \\
\qquad\qquad \text{if } (\mu = \mathbb{S}) \vee (\mu = \square) & & \\
\texttt{cost}_{\texttt{conv}}(\tau_1 \times \tau_2, \tau_1' \times \tau_2') & = & 2 \mathbin{\hat{+}} \texttt{cost}_{\texttt{conv}}(\tau_1, \tau_1') \mathbin{\hat{+}} \texttt{cost}_{\texttt{conv}}(\tau_2, \tau_2') \\
\texttt{cost}_{\texttt{conv}}(\texttt{unit}, \texttt{unit}) & = & 0 \\
\texttt{cost}_{\texttt{conv}}(\exists i.\ \tau, \exists i.\ \tau') & = & \texttt{cost}_{\texttt{conv}}(\tau, \tau') \\
\texttt{cost}_{\texttt{conv}}(C \to \tau, C \to \tau') & = & \texttt{cost}_{\texttt{conv}}(\tau, \tau') \\
\texttt{cost}_{\texttt{conv}}(C \wedge \tau, C \wedge \tau') & = & \texttt{cost}_{\texttt{conv}}(\tau, \tau') \\
\texttt{cost}_{\texttt{conv}}(\texttt{list}\,[n]^{\alpha}\ \tau, \texttt{list}\,[n]^{\alpha}\ \tau') & = & \alpha \mathbin{\hat{\cdot}} \texttt{cost}_{\texttt{conv}}(\tau^{\downarrow\square}, \tau'^{\downarrow\square}) \mathbin{\hat{+}} (n \mathbin{\hat{-}} \alpha) \mathbin{\hat{\cdot}} \texttt{cost}_{\texttt{conv}}(\tau, \tau')
\end{array}
$$

Figure 24: The cost associated with $\underline{\mathbf{conv}}(\cdot,\ \cdot,\ \cdot)$

$\boxed{\Delta; \Phi_A \vdash^{\kappa}_{\epsilon} A_1 \sqsubseteq A_2 \hookrightarrow \underline{\mathbf{g}}}$ A_1 is a subtype of A_2 and $\underline{\mathbf{g}}$ is the coercion metafunction

$\boxed{\Delta; \Phi_A \vdash^{\kappa}_{\epsilon} \tau_1 \sqsubseteq \tau_2 \hookrightarrow \underline{\mathbf{g}}}$ τ_1 is a subtype of τ_2 and $\underline{\mathbf{g}}$ is the coercion metafunction

The rules marked with (†) are also valid for the ${}_A\vdash$ judgment.

$$\frac{}{\Delta; \Phi \vdash^{0}_{\epsilon} (\texttt{int})^{\mathbb{S}} \sqsubseteq (\texttt{int})^{\square} \hookrightarrow \underline{\lambda} e.e} \qquad \frac{}{\Delta; \Phi \vdash^{0}_{\epsilon} (\texttt{int})^{\square} \sqsubseteq (\texttt{int})^{\mathbb{S}} \hookrightarrow \underline{\lambda} e.e}$$

$$\frac{\Delta; \Phi \vdash^{\kappa_1}_{\delta} \tau_1' \sqsubseteq \tau_1 \hookrightarrow \underline{\mathbf{g_1}} \qquad \Delta; \Phi \vdash^{\kappa_2}_{\delta} \tau_2 \sqsubseteq \tau_2' \hookrightarrow \underline{\mathbf{g_2}} \qquad \Delta; \Phi \models \kappa \mathbin{\hat{+}} \kappa_1 \mathbin{\hat{+}} \kappa_2 \mathbin{\hat{+}} (\delta = \mathbb{C})\ ?\ 1 :\ 0 \leq \kappa'}{\Delta; \Phi_A \vdash^{0}_{\epsilon} \tau_1 \xrightarrow{\delta(\kappa)} \tau_2 \sqsubseteq \tau_1' \xrightarrow{\delta(\kappa')} \tau_2' \hookrightarrow \underline{\lambda} e.\texttt{fix}\ f(x).\, \underline{\mathbf{g_2}}(e\ \underline{\mathbf{g_1}}(x))} \rightarrow 2$$

$$\frac{\Delta t :: S; \Phi \vdash^{\kappa_1}_{\epsilon} \tau \sqsubseteq \tau' \hookrightarrow \underline{\mathbf{g}} \qquad \Delta; \Phi \models \kappa \mathbin{\hat{+}} \kappa_1 \mathbin{\hat{+}} ((\delta = \mathbb{C})\ ?\ 1 :\ 0) \leq \kappa'}{\Delta; \Phi \vdash^{0}_{\epsilon} \forall t \overset{\delta(\kappa)}{::} S.\ \tau \sqsubseteq \forall t \overset{\delta(\kappa')}{::} S.\ \tau' \hookrightarrow \underline{\lambda} e.\lambda x.\, \underline{\mathbf{g}}(e\ ())} \forall 2$$

$$\frac{\Delta; \Phi \vdash^{\kappa_1}_{\epsilon} \tau_1 \sqsubseteq \tau_1' \hookrightarrow \underline{\mathbf{g_1}} \qquad \Delta; \Phi \vdash^{\kappa_2}_{\epsilon} \tau_2 \sqsubseteq \tau_2' \hookrightarrow \underline{\mathbf{g_2}} \qquad \Delta; \Phi \models \kappa \doteq \kappa_1 \mathbin{\hat{+}} \kappa_2}{\Delta; \Phi \vdash^{\kappa}_{\epsilon} \tau_1 \times \tau_2 \sqsubseteq \tau_1' \times \tau_2' \hookrightarrow \underline{\lambda} e.(\underline{\mathbf{g_1}}(\texttt{fst}\ e), \underline{\mathbf{g_2}}(\texttt{snd}\ e))} \times$$

$$\frac{\Delta; \Phi \models \kappa \doteq (\epsilon = \mathbb{C})\ ?\ 1 \mathbin{\hat{+}} \hat{\texttt{max}}(\texttt{cost}_{\texttt{conv}}(\tau_1, \tau_1^{\downarrow\square}), \texttt{cost}_{\texttt{conv}}(\tau_2, \tau_2^{\downarrow\square})) :\ 0}{\Delta; \Phi \vdash^{\kappa}_{\epsilon} (\tau_1 + \tau_2)^{\square} \sqsubseteq (\tau_1^{\downarrow\square} + \tau_2^{\downarrow\square})^{\square} \hookrightarrow \underline{\lambda} e.\texttt{case}(e, x.\texttt{inl}\ \underline{\mathbf{conv}}(x,\ \tau_1,\ \tau_1^{\downarrow\square}), y.\texttt{inr}\ \underline{\mathbf{conv}}(y,\ \tau_2,\ \tau_2^{\downarrow\square}))} +\mathbf{1}$$

$$\frac{\Delta; \Phi \models \kappa \doteq (\epsilon = \mathbb{C})\ ?\ 1 \mathbin{\hat{+}} \hat{\texttt{max}}(\texttt{cost}_{\texttt{conv}}(\tau_1^{\downarrow\square}, \tau_1), \texttt{cost}_{\texttt{conv}}(\tau_2^{\downarrow\square}, \tau_2)) :\ 0}{\Delta; \Phi \vdash^{\kappa}_{\epsilon} (\tau_1^{\downarrow\square} + \tau_2^{\downarrow\square})^{\square} \sqsubseteq (\tau_1 + \tau_2)^{\square} \hookrightarrow \underline{\lambda} e.\texttt{case}(e, x.\underline{\mathbf{conv}}(x,\ \tau_1^{\downarrow\square},\ \tau_1), y.\underline{\mathbf{conv}}(y,\ \tau_2^{\downarrow\square},\ \tau_2))} +\mathbf{2}$$

$$\frac{\Delta; \Phi \models \kappa \mathbin{\hat{+}} \texttt{cost}_{\texttt{conv}}(\tau_1^{\downarrow\square}, \tau_1) \mathbin{\hat{+}} (\delta = \mathbb{C})\ ?\ (1 \mathbin{\hat{+}} \texttt{cost}_{\texttt{conv}}(\tau_2, \tau_2^{\downarrow\square})) :\ 0 \mathrel{\dot{\leq}} \kappa'}{\Delta; \Phi \vdash^{0}_{\epsilon} (\tau_1 \xrightarrow{\delta(\kappa)} \tau_2)^{\square} \sqsubseteq (\tau_1^{\downarrow\square} \xrightarrow{\delta(\kappa')} \tau_2^{\downarrow\square})^{\square} \hookrightarrow \underline{\lambda} e.\lambda x.\, \underline{\mathbf{conv}}(e\ \underline{\mathbf{conv}}(x,\ \tau_1^{\downarrow\square},\ \tau_1),\ \tau_2,\ \tau_2^{\downarrow\square})} \rightarrow 1$$

$$\frac{\Delta; \Phi \models \kappa \mathbin{\hat{+}} (\delta = \mathbb{C})\ ?\ 1 \mathbin{\hat{+}} \texttt{cost}_{\texttt{conv}}(\tau^{\downarrow\square}, \tau) :\ 0 \mathrel{\dot{\leq}} \kappa'}{\Delta; \Phi \vdash^{0}_{\epsilon} (\forall t \overset{\delta(\kappa)}{::} S.\ \tau)^{\square} \sqsubseteq (\forall t \overset{\delta(\kappa')}{::} S.\ (\tau^{\downarrow\square}))^{\square} \hookrightarrow \underline{\lambda} e.\texttt{fix}\ f(x).\, \underline{\mathbf{conv}}(e\ (),\ \tau,\ \tau^{\downarrow\square})} \forall 1$$

$$\frac{\Delta; \Phi \models \alpha\ \doteq\ 0}{\Delta; \Phi \vdash^{0}_{\epsilon} \texttt{list}\,[n]^{\alpha}\ \tau \sqsubseteq \texttt{list}\,[n]^{\alpha}\ \tau^{\downarrow\square} \hookrightarrow \underline{\lambda} e.e} \ \mathbf{l1}$$

$$\frac{\Delta; \Phi \models n \doteq n' \qquad \Delta; \Phi \models \alpha \doteq \alpha' \qquad \Delta; \Phi \vdash^{\kappa'}_{\mathbb{S}} \tau \sqsubseteq \tau' \hookrightarrow \underline{\mathbf{g}} \qquad \Delta; \Phi \models \kappa \doteq \alpha \mathbin{\hat{\cdot}} \kappa'}{\Delta; \Phi \vdash^{\kappa}_{\mathbb{S}} \texttt{list}\,[n]^{\alpha}\ \tau \sqsubseteq \texttt{list}\,[n']^{\alpha'}\ \tau' \hookrightarrow \underline{\lambda} e.\texttt{map}\ \lambda z.\, \texttt{case}(z, x.\texttt{inl}\ \underline{\mathbf{conv}}(\texttt{drop}(\underline{\mathbf{g}}(\underline{\mathbf{conv}}(x,\ \tau^{\downarrow\square},\ \tau))),\ \tau',\ \tau'^{\downarrow\square}), y.\texttt{inr}\ \underline{\mathbf{g}}\ y)\ e} \ \mathbf{l2\text{-}\mathbb{S}}$$

$$\frac{\begin{array}{c}\Delta; \Phi \models n \doteq n' \qquad \Delta; \Phi \models \alpha \doteq \alpha' \\ \Delta; \Phi \vdash^{\kappa'}_{\mathbb{C}} \tau \sqsubseteq \tau' \hookrightarrow \underline{\mathbf{g}} \qquad \Delta; \Phi \models \kappa \doteq (n \mathbin{\hat{\cdot}} \kappa' \mathbin{\hat{+}} (n \mathbin{\hat{-}} \alpha) \mathbin{\hat{\cdot}} (\texttt{cost}_{\texttt{conv}}(\tau^{\downarrow\square}, \tau) \mathbin{\hat{+}} \texttt{cost}_{\texttt{conv}}(\tau', \tau'^{\downarrow\square}) \mathbin{\hat{+}} 1))\end{array}}{\Delta; \Phi \vdash^{\kappa}_{\mathbb{C}} \texttt{list}\,[n]^{\alpha}\ \tau \sqsubseteq \texttt{list}\,[n']^{\alpha'}\ \tau' \hookrightarrow \underline{\lambda} e.\texttt{map}\ \lambda z.\, \texttt{case}(z, x.\texttt{inl}\ \underline{\mathbf{conv}}(\underline{\mathbf{g}}(\underline{\mathbf{conv}}(x,\ \tau^{\downarrow\square},\ \tau)),\ \tau',\ \tau'^{\downarrow\square}), y.\texttt{inr}\ \underline{\mathbf{g}}\ y)\ e} \ \mathbf{l2\text{-}\mathbb{C}}$$

$$\frac{\Delta, t :: S; \Phi \vdash^{\kappa}_{\epsilon} \tau \sqsubseteq \tau' \hookrightarrow \underline{\mathbf{g}} \qquad t \notin FV(\Phi)}{\Delta; \Phi \vdash^{\kappa}_{\epsilon} \exists t.\ \tau \sqsubseteq \exists t.\ \tau' \hookrightarrow \underline{\mathbf{g}}} \ \exists \qquad \frac{\Delta; \Phi \models \mu \leq \mu' \qquad \Delta; \Phi \models \kappa \doteq \texttt{cost}_{\texttt{conv}}((A)^{\mu}, (A)^{\mu'})}{\Delta; \Phi \vdash^{\kappa}_{\epsilon} (A)^{\mu} \sqsubseteq (A)^{\mu'} \hookrightarrow \underline{\lambda} e.\underline{\mathbf{conv}}(e,\ (A)^{\mu},\ (A)^{\mu'})} \ \mu$$

$$\frac{\Delta; \Phi_A \vdash^{\kappa}_{\epsilon} A \sqsubseteq A' \hookrightarrow \underline{\mathbf{g}} \qquad \mu = \mathbb{S} \vee \mu = \square}{\Delta; \Phi \vdash^{\kappa}_{\epsilon} (A)^{\mu} \sqsubseteq (A')^{\mu} \hookrightarrow \underline{\mathbf{g}}} \ \mathbf{C1}$$

$$\frac{\Delta; \Phi_A \vdash^{\kappa'}_{\mathbb{S}} A \sqsubseteq A' \hookrightarrow \underline{\mathbf{g}} \qquad \Delta; \Phi \models \kappa \doteq \kappa' \mathbin{\hat{+}} 2}{\Delta; \Phi \vdash^{\kappa}_{\mathbb{S}} (A)^{\mathbb{C}} \sqsubseteq (A')^{\mathbb{C}} \hookrightarrow \underline{\lambda} e.\texttt{let}\ l\ =\ \texttt{ref}\ \underline{\mathbf{g}}(!e)\ \texttt{in}\ \texttt{let}\ ()\ =\ \texttt{push}([l],\ \lambda().\,\texttt{ref}\ \underline{\mathbf{g}}(!e))\ \texttt{in}\ l} \ \mathbf{C2\text{-}\mathbb{S}}$$

$$\frac{\Delta; \Phi_A \vdash^{\kappa'}_{\mathbb{C}} A \sqsubseteq A' \hookrightarrow \underline{\mathbf{g}} \qquad \Delta; \Phi \models \kappa \doteq \kappa' \mathbin{\hat{+}} 2}{\Delta; \Phi \vdash^{\kappa}_{\mathbb{C}} (A)^{\mathbb{C}} \sqsubseteq (A')^{\mathbb{C}} \hookrightarrow \underline{\lambda} e.\texttt{ref}\ \underline{\mathbf{g}}(!e)} \ \mathbf{C2\text{-}\mathbb{C}} \qquad \frac{}{\Delta; \Phi \vdash^{0}_{\epsilon} \tau \sqsubseteq \tau \hookrightarrow \underline{\lambda} e.e} \mathbf{refl*}(\dagger)$$

$$\frac{\Delta; \Phi \vdash^{\kappa_1}_{\epsilon} \tau_1 \sqsubseteq \tau_2 \hookrightarrow \underline{\mathbf{g_1}} \qquad \Delta; \Phi \vdash^{\kappa_2}_{\epsilon} \tau_2 \sqsubseteq \tau_3 \hookrightarrow \underline{\mathbf{g_2}} \qquad \Delta; \Phi \models \kappa \doteq \kappa_1 \mathbin{\hat{+}} \kappa_2}{\Delta; \Phi \vdash^{\kappa}_{\epsilon} \tau_1 \sqsubseteq \tau_3 \hookrightarrow \underline{\lambda} e.\underline{\mathbf{g_2}}(\underline{\mathbf{g_1}}(e))} \mathbf{tran}$$

$$\frac{\Delta; \Phi \wedge C \vdash^{\kappa_1}_{\epsilon} \eta \hookrightarrow \underline{\mathbf{g_1}} \qquad \Delta; \Phi \wedge \dot{\neg} C \vdash^{\kappa_2}_{\epsilon} \eta \hookrightarrow \underline{\mathbf{g_2}}}{\Delta; \Phi \vdash^{0}_{\epsilon} \eta \hookrightarrow \underline{\lambda} e.\bot} \mathbf{split}(\dagger) \qquad \frac{\Delta; \Phi \wedge C' \models C \qquad \Delta; \Phi \vdash^{\kappa}_{\epsilon} \tau \sqsubseteq \tau' \hookrightarrow \underline{\mathbf{g}}}{\Delta; \Phi \vdash^{\kappa}_{\epsilon} C \rightarrow \tau \sqsubseteq C' \rightarrow \tau' \hookrightarrow \underline{\mathbf{g}}} \mathbf{c\text{-}imp}$$

$$\frac{\Delta; \Phi \wedge C \models C' \qquad \Delta; \Phi \vdash^{\kappa}_{\epsilon} \tau \sqsubseteq \tau' \hookrightarrow \underline{\mathbf{g}}}{\Delta; \Phi \vdash^{\kappa}_{\epsilon} C \wedge \tau \sqsubseteq C' \wedge \tau' \hookrightarrow \underline{\mathbf{g}}} \mathbf{c\text{-}and}$$

Figure 25: Coercion meta-function

$$\boxed{\Delta; \Phi; \Gamma \vdash^{\kappa}_{\epsilon} e : \tau \hookrightarrow \ulcorner e \urcorner}$$

$$\frac{\Delta; \Phi \vdash \Gamma \ \mathtt{wf}}{\Delta; \Phi; \Gamma, x : \tau \vdash^{0}_{\epsilon} x : \tau \hookrightarrow x}\ \textbf{var} \qquad \frac{\Delta; \Phi \vdash \Gamma \ \mathtt{wf}}{\Delta; \Phi; \Gamma \vdash^{0}_{\epsilon} \mathtt{n} : (\mathtt{int})^{\square} \hookrightarrow \mathtt{n}}\ \textbf{int}$$

$$\frac{\Delta; \Phi; \Gamma \vdash^{\kappa_1}_{\epsilon} e_1 : \tau_1 \hookrightarrow \ulcorner e_1 \urcorner \qquad \Delta; \Phi; \Gamma \vdash^{\kappa_2}_{\epsilon} e_1 : \tau_2 \hookrightarrow \ulcorner e_2 \urcorner \qquad \Delta; \Phi \models \kappa \doteq \kappa_1 \mathbin{\hat{+}} \kappa_2}{\Delta; \Phi; \Gamma \vdash^{\kappa}_{\epsilon} (e_1, e_2) : \tau_1 \times \tau_2 \hookrightarrow (\ulcorner e_1 \urcorner, \ulcorner e_2 \urcorner)}\ \textbf{pair}$$

$$\frac{\Delta; \Phi; \Gamma \vdash^{\kappa}_{\epsilon} e : \tau_1 \hookrightarrow \ulcorner e \urcorner}{\Delta; \Phi; \Gamma \vdash^{\kappa}_{\epsilon} \mathtt{inl}\ e : (\tau_1 + \tau_2)^{\mathbb{S}} \hookrightarrow \mathtt{inl}\ \ulcorner e \urcorner}\ \textbf{inl} \qquad \frac{\Delta; \Phi; \Gamma \vdash^{\kappa}_{\epsilon} e : \tau_2 \hookrightarrow \ulcorner e \urcorner}{\Delta; \Phi; \Gamma \vdash^{\kappa}_{\epsilon} \mathtt{inr}\ e : (\tau_1 + \tau_2)^{\mathbb{S}} \hookrightarrow \mathtt{inr}\ \ulcorner e \urcorner}\ \textbf{inr}$$

$$\frac{\Delta; \Phi; \Gamma \vdash^{\kappa'}_{\epsilon} e : \tau_1 \times \tau_2 \hookrightarrow \ulcorner e \urcorner \qquad \Delta; \Phi \models \kappa \doteq (\kappa' \mathbin{\hat{+}} \epsilon = \mathbb{C}\ ?\ 1\ :\ 0)}{\Delta; \Phi; \Gamma \vdash^{\kappa}_{\epsilon} \mathtt{fst}\ e : \tau_1 \hookrightarrow \mathtt{fst}\ \ulcorner e \urcorner}\ \textbf{fst}$$

$$\frac{\Delta; \Phi; \Gamma \vdash^{\kappa'}_{\epsilon} e : \tau_1 \times \tau_2 \hookrightarrow \ulcorner e \urcorner \qquad \Delta; \Phi \models \kappa \doteq (\kappa' \mathbin{\hat{+}} \epsilon = \mathbb{C}\ ?\ 1\ :\ 0)}{\Delta; \Phi; \Gamma \vdash^{\kappa}_{\epsilon} \mathtt{snd}\ e : \tau_2 \hookrightarrow \mathtt{fst}\ \ulcorner e \urcorner}\ \textbf{snd}$$

$$\frac{\Delta; \Phi; \Gamma, f : (\tau_1 \xrightarrow{\delta(\kappa)} \tau_2)^{\mathbb{S}}, x : \tau_1 \vdash^{\kappa}_{\delta} e : \tau_2 \hookrightarrow \ulcorner e \urcorner}{\Delta; \Phi; \Gamma \vdash^{0}_{\epsilon} \mathtt{fix}\ f(x).\, e : (\tau_1 \xrightarrow{\delta(\kappa)} \tau_2)^{\mathbb{S}} \hookrightarrow \mathtt{fix}\ f(x).\, \ulcorner e \urcorner}\ \textbf{fix1}$$

$$\frac{\Delta; \Phi; \Gamma \vdash^{\kappa_1}_{\mathbb{S}} e_1 : (\tau_1 \xrightarrow{\mathbb{S}(\kappa')} \tau_2)^{\mu} \hookrightarrow \ulcorner e_1 \urcorner \qquad \Delta; \Phi; \Gamma \vdash^{\kappa_2}_{\mathbb{S}} e_2 : \tau_1 \hookrightarrow \ulcorner e_2 \urcorner \qquad \mu \leq \mathbb{S} \qquad \Delta; \Phi \models \kappa \doteq \kappa' \mathbin{\hat{+}} \kappa_1 \mathbin{\hat{+}} \kappa_2}{\Delta; \Phi; \Gamma \vdash^{\kappa}_{\mathbb{S}} e_1\ e_2 : \tau_2 \hookrightarrow \ulcorner e_1 \urcorner\, \ulcorner e_2 \urcorner}\ \textbf{app1}$$

$$\frac{\begin{array}{c} \Delta; \Phi; \Gamma \vdash^{\kappa_1}_{\mathbb{S}} e_1 : (\tau_1 \xrightarrow{\mathbb{C}(\kappa')} \tau_2)^{\mathbb{C}} \hookrightarrow \ulcorner e_1 \urcorner \\ \Delta; \Phi; \Gamma \vdash^{\kappa_2}_{\mathbb{S}} e_2 : \tau_1 \hookrightarrow \ulcorner e_2 \urcorner \qquad \mathbb{C} \trianglelefteq \tau_2 \qquad \Delta; \Phi \models \kappa \doteq \kappa' \mathbin{\hat{+}} \kappa_1 \mathbin{\hat{+}} \kappa_2 \mathbin{\hat{+}} \mathtt{cost}_{\mathtt{deepref}}(\tau_2) \mathbin{\hat{+}} 3 \end{array}}{\Delta; \Phi; \Gamma \vdash^{\kappa}_{\mathbb{S}} e_1\ e_2 : \tau_2 \hookrightarrow \begin{array}{l} \mathtt{let}\ l\ =\ \ulcorner e_1 \urcorner\ \mathtt{in}\ \mathtt{let}\ x\ =\ \ulcorner e_2 \urcorner\ \mathtt{in}\ \mathtt{let}\ r\ =\ \underline{\mathbf{deep}}(!l\ x,\ \tau_2)\ \mathtt{in} \\ \mathtt{let}\ ()\ =\ \mathtt{push}(\mathtt{snd}\ r,\ \lambda().\, \underline{\mathbf{deep}'}(!l\ x,\ \tau_2))\ \mathtt{in}\ \mathtt{fst}\ r \end{array}}\ \textbf{app2}$$

$$\frac{\begin{array}{c} \Delta; \Phi; \Gamma \vdash^{\kappa_1}_{\mathbb{S}} e_1 : (\tau_1 \xrightarrow{\mathbb{C}(\kappa')} \tau_2)^{\mu} \hookrightarrow \ulcorner e_1 \urcorner \\ \Delta; \Phi; \Gamma \vdash^{\kappa_2}_{\mathbb{S}} e_2 : \tau_1 \hookrightarrow \ulcorner e_2 \urcorner \qquad \mathbb{C} \trianglelefteq \tau_2 \qquad \mu \leq \mathbb{S} \qquad \Delta; \Phi \models \kappa \doteq \kappa' \mathbin{\hat{+}} \kappa_1 \mathbin{\hat{+}} \kappa_2 \mathbin{\hat{+}} \mathtt{cost}_{\mathtt{deepref}}(\tau_2) \mathbin{\hat{+}} 2 \end{array}}{\Delta; \Phi; \Gamma \vdash^{\kappa}_{\mathbb{S}} e_1\ e_2 : \tau_2 \hookrightarrow \begin{array}{l} \mathtt{let}\ f\ =\ \ulcorner e_1 \urcorner\ \mathtt{in}\ \mathtt{let}\ x\ =\ \ulcorner e_2 \urcorner\ \mathtt{in}\ \mathtt{let}\ r\ =\ \underline{\mathbf{deep}}(f\ x,\ \tau_2)\ \mathtt{in} \\ \mathtt{let}\ ()\ =\ \mathtt{push}(\mathtt{snd}\ r,\ \lambda().\, \underline{\mathbf{deep}'}(f\ x,\ \tau_2))\ \mathtt{in}\ \mathtt{fst}\ r \end{array}}\ \textbf{app3}$$

$$\frac{\begin{array}{c} \Delta; \Phi; \Gamma \vdash^{\kappa_1}_{\mathbb{C}} e_1 : (\tau_1 \xrightarrow{\mathbb{C}(\kappa')} \tau_2)^{\mu} \hookrightarrow \ulcorner e_1 \urcorner \\ \Delta; \Phi; \Gamma \vdash^{\kappa_2}_{\mathbb{C}} e_2 : \tau_1 \hookrightarrow \ulcorner e_2 \urcorner \qquad \mathbb{C} \trianglelefteq \tau_2 \qquad \Delta; \Phi \models \kappa \doteq \kappa' \mathbin{\hat{+}} \kappa_1 \mathbin{\hat{+}} kappa_2 \mathbin{\hat{+}} 2 \end{array}}{\Delta; \Phi; \Gamma \vdash^{\kappa}_{\mathbb{C}} e_1\ e_2 : \tau_2 \hookrightarrow\ !\ulcorner e_1 \urcorner\, \ulcorner e_2 \urcorner}\ \textbf{app4}$$

$$\frac{\Delta; \Phi; \Gamma \vdash^{\kappa_1}_{\mathbb{C}} e_1 : (\tau_1 \xrightarrow{\mathbb{C}(\kappa')} \tau_2)^{\mathbb{S}} \hookrightarrow \ulcorner e_1 \urcorner \qquad \Delta; \Phi; \Gamma \vdash^{\kappa_2}_{\mathbb{C}} e_2 : \tau_1 \hookrightarrow \ulcorner e_2 \urcorner \qquad \mathbb{C} \trianglelefteq \tau_2 \qquad \Delta; \Phi \models \kappa \doteq \kappa' \mathbin{\hat{+}} \kappa_1 \mathbin{\hat{+}} \kappa_2 \mathbin{\hat{+}} 1}{\Delta; \Phi; \Gamma \vdash^{\kappa}_{\mathbb{C}} e_1\ e_2 : \tau_2 \hookrightarrow \ulcorner e_1 \urcorner\, \ulcorner e_2 \urcorner}\ \textbf{app5}$$

$$\frac{\begin{array}{c} \Delta; \Phi; \Gamma \vdash^{\kappa_e}_{\epsilon} e : (\tau_1 + \tau_2)^{\mu} \hookrightarrow \ulcorner e \urcorner \qquad \Delta; \Phi; \Gamma, x : \tau_1 \vdash^{\kappa'}_{\epsilon} e_1 : \tau \hookrightarrow \ulcorner e_1 \urcorner \\ \Delta; \Phi; \Gamma, y : \tau_2 \vdash^{\kappa'}_{\epsilon} e_2 : \tau \hookrightarrow \ulcorner e_2 \urcorner \qquad \mu \leq \mathbb{S} \qquad \Delta; \Phi \models \kappa \doteq \kappa_e \mathbin{\hat{+}} \kappa' \mathbin{\hat{+}} ((\epsilon = \mathbb{C})\ ?\ 1\ :\ 0) \end{array}}{\Delta; \Phi; \Gamma \vdash^{\kappa}_{\epsilon} \mathtt{case}(e, x.e_1, y.e_2) : \tau \hookrightarrow \mathtt{case}(\ulcorner e \urcorner, x.\ulcorner e_1 \urcorner, y.\ulcorner e_2 \urcorner)}\ \textbf{case1}$$

$$\frac{\begin{array}{c} \Delta; \Phi; \Gamma \vdash^{\kappa_e}_{\mathbb{S}} e : (\tau_1 + \tau_2)^{\mathbb{C}} \hookrightarrow \ulcorner e \urcorner \qquad \Delta; \Phi; \Gamma, x : \tau_1 \vdash^{\kappa'}_{\mathbb{C}} e_1 : \tau \hookrightarrow \ulcorner e_1 \urcorner \\ \Delta; \Phi; \Gamma, y : \tau_2 \vdash^{\kappa'}_{\mathbb{C}} e_2 : \tau \hookrightarrow \ulcorner e_2 \urcorner \qquad \mathbb{C} \trianglelefteq \tau \qquad \Delta; \Phi \models \kappa \doteq \kappa_e \mathbin{\hat{+}} \kappa' \mathbin{\hat{+}} \mathtt{cost}_{\mathtt{deepref}}(\tau) \mathbin{\hat{+}} 3 \end{array}}{\Delta; \Phi; \Gamma \vdash^{\kappa}_{\mathbb{S}} \mathtt{case}(e, x.e_1, y.e_2) : \tau \hookrightarrow \begin{array}{l} \mathtt{let}\ l\ =\ \ulcorner e \urcorner\ \mathtt{in}\ \mathtt{let}\ r\ =\ \underline{\mathbf{deep}}(\mathtt{case}(!l, x.\ulcorner e_1 \urcorner, y.\ulcorner e_2 \urcorner),\ \tau)\ \mathtt{in} \\ \mathtt{let}\ ()\ =\ \mathtt{push}(\mathtt{snd}\ r,\ \lambda().\, \underline{\mathbf{deep}'}(\mathtt{case}(!l, x.\ulcorner e_1 \urcorner, y.\ulcorner e_2 \urcorner),\ \tau))\ \mathtt{in} \\ \mathtt{fst}\ r \end{array}}\ \textbf{case2}$$

Figure 26: Translation rules

$\boxed{\Delta; \Phi; \Gamma \vdash^{\kappa}_{\epsilon} e : \tau}$ expression e has type τ with dynamic stability κ. The context Υ carrying types of primitive functions is omitted from all rules.

$$\frac{\begin{array}{c}\Delta; \Phi; \Gamma \vdash^{\kappa_e}_{\epsilon} e : (\tau_1 + \tau_2)^{\mathbb{C}} \hookrightarrow \ulcorner e \urcorner \\ \Delta; \Phi; \Gamma, x : \tau_1 \vdash^{\kappa'}_{\mathbb{C}} e_1 : \tau \hookrightarrow \ulcorner e_1 \urcorner \qquad \Delta; \Phi; \Gamma, y : \tau_2 \vdash^{\kappa'}_{\mathbb{C}} e_2 : \tau \hookrightarrow \ulcorner e_2 \urcorner \qquad \mu \trianglelefteq \tau \qquad \Delta; \Phi \models \kappa \doteq \kappa_e \mathbin{\hat{+}} \kappa' \mathbin{\hat{+}} 2\end{array}}{\Delta; \Phi; \Gamma \vdash^{\kappa}_{\mathbb{C}} \texttt{case}(e, x.e_1, y.e_2) : \tau \hookrightarrow \texttt{case}(!\ulcorner e \urcorner, x.\ulcorner e_1 \urcorner, y.\ulcorner e_2 \urcorner)}\ \textbf{case3}$$

$$\frac{\begin{array}{c}\Upsilon(\zeta) = \zeta : (T_1, \ldots, T_n) \xrightarrow{\kappa'} T \\ \Delta; \Phi; \Gamma \vdash^{\kappa_{e_i}}_{\epsilon} e_i : (T_i)^{\mu_i} \hookrightarrow \ulcorner e_i \urcorner \qquad \mu_1 \sqcup \cdots \sqcup \mu_n = \mathbb{S} \qquad \Delta; \Phi \models \kappa \doteq (\sum_1^n \kappa_{e_i}) \mathbin{\hat{+}} (\epsilon \doteq \mathbb{C}\ ?\ \kappa' \mathbin{\hat{+}} 1 :\ 0)\end{array}}{\Delta; \Phi; \Gamma \vdash^{\kappa}_{\epsilon} \zeta(e_1, \ldots, e_n) : (T)^{\mu_1 \sqcup \cdots \sqcup \mu_n} \hookrightarrow \zeta(\ulcorner e_1 \urcorner, \ldots, \ulcorner e_n \urcorner)}\ \textbf{primApp1}$$

$$\frac{\begin{array}{c}\Upsilon(\zeta) = \zeta : (T_1, \ldots, T_n) \xrightarrow{\kappa'} T \qquad \Delta; \Phi; \Gamma \vdash^{\kappa_{e_i}}_{\mathbb{S}} e_i : (T_i)^{\mu_i} \hookrightarrow \ulcorner e_i \urcorner \\ \mu_1 \sqcup \cdots \sqcup \mu_n = \mathbb{C} \qquad \Delta; \Phi \models \kappa \doteq (\sum_1^n \kappa_{e_i}) \mathbin{\hat{+}} \kappa' \mathbin{\hat{+}} 2 \qquad x'_i = \texttt{if } (\mu_i = \mathbb{C}) \texttt{ then } !x_i \texttt{ else } x_i\end{array}}{\Delta; \Phi; \Gamma \vdash^{\kappa}_{\mathbb{S}} \zeta(e_1, \ldots, e_n) : (T)^{\mu_1 \sqcup \cdots \sqcup \mu_n} \hookrightarrow \begin{array}{l}\texttt{let } x_i = \ulcorner e_i \urcorner \texttt{ in} \\ \texttt{let } l = \texttt{ref } (\zeta(x'_1, \ldots, x'_n)) \texttt{ in} \\ \texttt{let } () = \texttt{push}([l],\ \lambda().\,\texttt{ref } (\zeta(x'_1, \ldots, x'_n))) \texttt{ in } l\end{array}}\ \textbf{primApp2}$$

$$\frac{\begin{array}{c}\Upsilon(\zeta) = \zeta : (T_1, \ldots, T_n) \xrightarrow{\kappa'} T \qquad \Delta; \Phi; \Gamma \vdash^{\kappa_{e_i}}_{\mathbb{C}} e_i : (T_i)^{\mu_i} \hookrightarrow \ulcorner e_i \urcorner \\ \mu_1 \sqcup \cdots \sqcup \mu_n = \mathbb{C} \qquad \Delta; \Phi \models \kappa \doteq (\sum_1^n \kappa_{e_i}) \mathbin{\hat{+}} \kappa' \mathbin{\hat{+}} 1 \qquad \ulcorner e'_i \urcorner = \texttt{if } (\mu_i = \mathbb{C}) \texttt{ then } !\ulcorner e_i \urcorner \texttt{ else } \ulcorner e_i \urcorner\end{array}}{\Delta; \Phi; \Gamma \vdash^{\kappa}_{\mathbb{C}} \zeta(e_1, \ldots, e_n) : (T)^{\mu_1 \sqcup \cdots \sqcup \mu_n} \hookrightarrow \zeta(\ulcorner e'_1 \urcorner, \ldots, \ulcorner e'_n \urcorner)}\ \textbf{primApp3}$$

$$\frac{\Delta, t :: S; \Phi; \Gamma \vdash^{\kappa}_{\delta} e : \tau \hookrightarrow \ulcorner e \urcorner}{\Delta; \Phi; \Gamma \vdash^{0}_{\epsilon} \Lambda.\, e : \forall t \overset{\delta(\kappa)}{::} S.\ \tau \hookrightarrow \lambda().\,\ulcorner e \urcorner}\ \forall\textbf{I}$$

$$\frac{\Delta; \Phi; \Gamma \vdash^{\kappa_e}_{\mathbb{S}} e : (\forall t \overset{\mathbb{S}(\kappa')}{::} S.\ \tau)^{\mathbb{S}} \hookrightarrow \ulcorner e \urcorner \qquad \Delta \vdash I :: S \qquad \Delta; \Phi \models \kappa \doteq \kappa_e \mathbin{\hat{+}} \kappa'\{I/t\}}{\Delta; \Phi; \Gamma \vdash^{\kappa}_{\mathbb{S}} e[\,] : \tau\{I/t\} \hookrightarrow \ulcorner e \urcorner\ ()}\ \forall\textbf{E1}$$

$$\frac{\begin{array}{c}\Delta; \Phi; \Gamma \vdash^{\kappa_e}_{\mathbb{S}} e : (\forall t \overset{\mathbb{C}(\kappa')}{::} S.\ \tau)^{\mathbb{C}} \hookrightarrow \ulcorner e \urcorner \\ \Delta \vdash I :: S \qquad \mathbb{C} \trianglelefteq \tau\{I/t\} \qquad \Delta; \Phi \models \kappa \doteq \kappa_e \mathbin{\hat{+}} \kappa'\{I/t\} \mathbin{\hat{+}} \texttt{cost}_{\texttt{deepref}}(\tau) \mathbin{\hat{+}} 3\end{array}}{\Delta; \Phi; \Gamma \vdash^{\kappa}_{\mathbb{S}} e[\,] : \tau\{I/t\} \hookrightarrow \begin{array}{l}\texttt{let } l = \ulcorner e \urcorner \texttt{ in let } r = \underline{\textbf{deep}}(!l\ (),\ \tau\{I/t\}) \texttt{ in} \\ \texttt{let } () = \texttt{push}(\texttt{snd } r,\ \lambda().\,\underline{\textbf{deep}'}(!l\ (),\ \tau\{I/t\})) \texttt{ in fst } r\end{array}}\ \forall\textbf{E2}$$

$$\frac{\begin{array}{c}\Delta; \Phi; \Gamma \vdash^{\kappa_e}_{\mathbb{S}} e : (\forall t \overset{\mathbb{C}(\kappa')}{::} S.\ \tau)^{\mathbb{S}} \hookrightarrow \ulcorner e \urcorner \\ \Delta \vdash I :: S \qquad \mathbb{C} \trianglelefteq \tau\{I/t\} \qquad \Delta; \Phi \models \kappa \doteq \kappa_e \mathbin{\hat{+}} \kappa'\{I/t\} \mathbin{\hat{+}} \texttt{cost}_{\texttt{deepref}}(\tau) \mathbin{\hat{+}} 2\end{array}}{\Delta; \Phi; \Gamma \vdash^{\kappa}_{\mathbb{S}} e[\,] : \tau\{I/t\} \hookrightarrow \begin{array}{l}\texttt{let } f = \ulcorner e \urcorner \texttt{ in let } r = \underline{\textbf{deep}}(f\ (),\ \tau\{I/t\}) \texttt{ in} \\ \texttt{let } () = \texttt{push}(\texttt{snd } r,\ \lambda().\,\underline{\textbf{deep}'}(f\ (),\ \tau\{I/t\})) \texttt{ in fst } r\end{array}}\ \forall\textbf{E3}$$

$$\frac{\Delta; \Phi; \Gamma \vdash^{\kappa_e}_{\mathbb{C}} e : (\forall t \overset{\mathbb{C}(\kappa')}{::} S.\ \tau)^{\mathbb{C}} \hookrightarrow \ulcorner e \urcorner \qquad \Delta \vdash I :: S \qquad \mathbb{C} \trianglelefteq \tau\{I/t\} \qquad \Delta; \Phi \models \kappa \doteq \kappa_e \mathbin{\hat{+}} \kappa'\{I/t\} \mathbin{\hat{+}} 2}{\Delta; \Phi; \Gamma \vdash^{\kappa}_{\mathbb{C}} e[\,] : \tau\{I/t\} \hookrightarrow !\ulcorner e \urcorner\ ()}\ \forall\textbf{E4}$$

$$\frac{\Delta; \Phi; \Gamma \vdash^{\kappa_e}_{\mathbb{C}} e : (\forall t \overset{\mathbb{C}(\kappa')}{::} S.\ \tau)^{\mathbb{S}} \hookrightarrow \ulcorner e \urcorner \qquad \Delta \vdash I :: S \qquad \mathbb{C} \trianglelefteq \tau\{I/t\} \qquad \Delta; \Phi \models \kappa \doteq \kappa_e \mathbin{\hat{+}} \kappa'\{I/t\} \mathbin{\hat{+}} 1}{\Delta; \Phi; \Gamma \vdash^{\kappa}_{\mathbb{C}} e[\,] : \tau\{I/t\} \hookrightarrow \ulcorner e \urcorner\ ()}\ \forall\textbf{E5}$$

$$\frac{\Delta; \Phi; \Gamma \vdash^{\kappa}_{\epsilon} e : \tau\{I/t\} \hookrightarrow \ulcorner e \urcorner \qquad \Delta \vdash I :: S}{\Delta; \Phi; \Gamma \vdash^{\kappa}_{\epsilon} \texttt{pack } e : \exists t.\ \tau \hookrightarrow \ulcorner e \urcorner}\ \exists\textbf{I}$$

$$\frac{\begin{array}{c}\Delta; \Phi; \Gamma \vdash^{\kappa_e}_{\epsilon} e : \exists t.\ \tau \hookrightarrow \ulcorner e \urcorner \\ \Delta, t :: S; \Phi; \Gamma, x : \tau \vdash^{\kappa'}_{\epsilon} e' : \tau' \hookrightarrow \ulcorner e' \urcorner \qquad t \notin FV(\Phi; \Gamma, \tau') \qquad \Delta; \Phi \models \kappa \doteq \kappa_e \mathbin{\hat{+}} \kappa' \mathbin{\hat{+}} ((\epsilon = \mathbb{C})\ ?\ 1 :\ 0)\end{array}}{\Delta; \Phi; \Gamma \vdash^{\kappa}_{\epsilon} \texttt{unpack } e \texttt{ as } x \texttt{ in } e' : \tau' \hookrightarrow \texttt{let } x = \ulcorner e \urcorner \texttt{ in } \ulcorner e' \urcorner}\ \exists\textbf{E}$$

Figure 27: Translation rules, part 2

$$\boxed{\Delta; \Phi; \Gamma \vdash^{\kappa}_{\epsilon} e : \tau \hookrightarrow \ulcorner e \urcorner}$$

$$\frac{\Delta; \Phi; \Gamma \vdash^{\kappa_1}_{\epsilon} e : \tau' \hookrightarrow \ulcorner e \urcorner \qquad \Delta; \Phi \vdash^{\kappa_2}_{\epsilon} \tau' \sqsubseteq \tau \hookrightarrow \underline{\mathbf{g}} \qquad \Delta; \Phi \models \kappa_1 \mathbin{\hat{+}} \kappa_2 \mathrel{\dot{\leq}} \kappa}{\Delta; \Phi; \Gamma \vdash^{\kappa}_{\epsilon} e : \tau \hookrightarrow \underline{\mathbf{g}}(\ulcorner e \urcorner)} \sqsubseteq$$

$$\frac{\Delta; \Phi; \Gamma \vdash^{\kappa'}_{\epsilon} e : \tau \hookrightarrow \ulcorner e \urcorner \qquad \forall x \in \Gamma \;\; \Delta; \Phi \vdash \Gamma(x) \sqsubseteq \Gamma(x)^{\downarrow\Box} \qquad \Delta; \Phi \models \kappa \doteq (\epsilon = \mathbb{S} \;?\; 0 \;:\; \kappa')}{\Delta; \Phi; \Gamma, \Gamma' \vdash^{\kappa}_{\epsilon} e : \tau^{\downarrow\Box} \hookrightarrow \mathtt{drop}(\underline{\mathbf{conv}}(\ulcorner e \urcorner,\ \tau,\ \tau^{\downarrow\Box}))} \;\textbf{nochange}$$

$$\frac{\Delta; \Phi; \Gamma, f : (\tau_1 \xrightarrow{\delta(\kappa)} \tau_2)^{\Box}, x : \tau_1 \vdash^{\kappa}_{\delta} e : \tau_2 \hookrightarrow \ulcorner e \urcorner \qquad \forall x \in \Gamma \;\; \Delta; \Phi \models \Gamma(x) \sqsubseteq \Gamma(x)^{\downarrow\Box}}{\Delta; \Phi; \Gamma, \Gamma' \vdash^{0}_{\epsilon} \mathtt{fix}\ f(x).\, e : (\tau_1 \xrightarrow{\delta(\kappa)} \tau_2)^{\Box} \hookrightarrow \mathtt{fix}\ f(x).\, \ulcorner e \urcorner} \;\textbf{fix2}$$

$$\frac{\Delta; \Phi; \Gamma \vdash^{\kappa_1}_{\epsilon} e_1 : \tau_1 \hookrightarrow \ulcorner e_1 \urcorner \qquad \Delta; \Phi; \Gamma, x : \tau_1 \vdash^{\kappa_2}_{\epsilon} e_2 : \tau_2 \hookrightarrow \ulcorner e_2 \urcorner \qquad \Delta; \Phi \models \kappa \doteq \kappa_1 + \kappa_2 + (\epsilon = \mathbb{C} \;?\; 1 \;:\; 0)}{\Delta; \Phi; \Gamma \vdash^{\kappa}_{\epsilon} \mathtt{let}\ x \ =\ e_1\ \mathtt{in}\ e_2 : \tau_2 \hookrightarrow \mathtt{let}\ x \ =\ \ulcorner e_1 \urcorner\ \mathtt{in}\ \ulcorner e_2 \urcorner} \;\textbf{let}$$

$$\frac{\Delta; \Phi \vdash \Gamma\ \mathtt{wf}}{\Delta; \Phi; \Gamma \vdash^{0}_{\epsilon} () : \mathtt{unit} \hookrightarrow ()} \;\textbf{unit} \qquad \frac{\Delta; \Phi \wedge C; \Gamma \vdash^{\kappa}_{\epsilon} e : \tau \hookrightarrow \ulcorner e \urcorner}{\Delta; \Phi; \Gamma \vdash^{\kappa}_{\epsilon} e : C \to \tau \hookrightarrow \ulcorner e \urcorner} \;\textbf{c-impI}$$

$$\frac{\Delta; \Phi; \Gamma \vdash^{\kappa}_{\epsilon} e : C \to \tau \hookrightarrow \ulcorner e \urcorner \qquad \Delta; \Phi \models C}{\Delta; \Phi; \Gamma \vdash^{\kappa}_{\epsilon} e : \tau \hookrightarrow \ulcorner e \urcorner} \;\textbf{c-impE} \qquad \frac{\Delta; \Phi; \Gamma \vdash^{\kappa}_{\epsilon} e : \tau \hookrightarrow \ulcorner e \urcorner \qquad \Delta; \Phi \models C}{\Delta; \Phi; \Gamma \vdash^{\kappa}_{\epsilon} e : C \wedge \tau \hookrightarrow \ulcorner e \urcorner} \;\textbf{c-andI}$$

$$\frac{\Delta; \Phi; \Gamma \vdash^{\kappa}_{\epsilon} e_1 : C \wedge \tau_1 \hookrightarrow \ulcorner e_1 \urcorner \qquad \Delta; \Phi \wedge C; \Gamma, x : \tau_1 \vdash^{\kappa}_{\epsilon} e_2 : \tau_2 \hookrightarrow \ulcorner e_2 \urcorner}{\Delta; \Phi; \Gamma \vdash^{\kappa}_{\epsilon} \mathtt{let}\ x \ =\ e_1\ \mathtt{in}\ e_2 : \tau_2 \hookrightarrow \mathtt{let}\ x \ =\ \ulcorner e_1 \urcorner\ \mathtt{in}\ \ulcorner e_2 \urcorner} \;\textbf{c-andE}$$

$$\frac{\Delta; \Phi \models \bot \qquad \Delta; \Phi \vdash \Gamma\ \mathtt{wf}}{\Delta; \Phi; \Gamma \vdash^{\kappa}_{\epsilon} e : \tau \hookrightarrow \bot} \;\textbf{contra} \qquad \frac{\Delta; \Phi \vdash \Gamma\ \mathtt{wf}}{\Delta; \Phi; \Gamma \vdash^{0}_{\epsilon} [] : \mathtt{list}\, [0]^{0}\ \tau \hookrightarrow []} \;\textbf{nil}$$

$$\frac{\Delta; \Phi; \Gamma \vdash^{\kappa_1}_{\epsilon} e_1 : \tau^{\downarrow\Box} \hookrightarrow \ulcorner e_1 \urcorner \qquad \Delta; \Phi; \Gamma \vdash^{\kappa_2}_{\epsilon} e_2 : \mathtt{list}\, [n]^{\alpha}\ \tau \hookrightarrow \ulcorner e_2 \urcorner \qquad \Delta; \Phi \models \kappa \doteq \kappa_1 \mathbin{\hat{+}} \kappa_2}{\Delta; \Phi; \Gamma \vdash^{\kappa}_{\epsilon} e_1 :: e_2 : \mathtt{list}\, \left[n \mathbin{\hat{+}} 1\right]^{\alpha}\ \tau \hookrightarrow \mathtt{inl}\ \ulcorner e_1 \urcorner :: \ulcorner e_2 \urcorner} \;\textbf{cons1}$$

$$\frac{\Delta; \Phi; \Gamma \vdash^{\kappa_1}_{\epsilon} e_1 : \tau \hookrightarrow \ulcorner e_1 \urcorner \qquad \Delta; \Phi; \Gamma \vdash^{\kappa_2}_{\epsilon} e_2 : \mathtt{list}\, [n]^{\alpha - 1}\ \tau \hookrightarrow \ulcorner e_2 \urcorner \qquad \Delta; \Phi \models \alpha \mathrel{\dot{>}} 0 \qquad \Delta; \Phi \models \kappa \doteq \kappa_1 \mathbin{\hat{+}} \kappa_2}{\Delta; \Phi; \Gamma \vdash^{\kappa}_{\epsilon} e_1 :: e_2 : \mathtt{list}\, \left[n \mathbin{\hat{+}} 1\right]^{\alpha}\ \tau \hookrightarrow \mathtt{inr}\ \ulcorner e_1 \urcorner :: \ulcorner e_2 \urcorner} \;\textbf{cons2}$$

$$\frac{\begin{array}{c} \Delta; \Phi; \Gamma \vdash^{\kappa_e}_{\epsilon} e : \mathtt{list}\, [n]^{\alpha}\ \tau \hookrightarrow \ulcorner e \urcorner \\ \Delta; \Phi \wedge n \ \doteq\ 0; \Gamma \vdash^{\kappa'}_{\epsilon} e_1 : \tau' \hookrightarrow \ulcorner e_1 \urcorner \qquad \Delta, i :: \iota; \Phi \wedge n \doteq i + 1; \Gamma, h : \tau^{\downarrow\Box}, tl : \mathtt{list}\, [i]^{\alpha}\ \tau \vdash^{\kappa'}_{\epsilon} e_2 : \tau' \hookrightarrow \ulcorner e_2 \urcorner_{l} \\ \Delta, i :: \iota, \beta :: \iota; \Phi \wedge n \doteq i + 1 \wedge \alpha \doteq \beta + 1; \Gamma, h : \tau, tl : \mathtt{list}\, [i]^{\beta}\ \tau \vdash^{\kappa'}_{\epsilon} e_2 : \tau' \hookrightarrow \ulcorner e_2 \urcorner_{r} \\ \Delta; \Phi \models \kappa \doteq \kappa_e \mathbin{\hat{+}} \kappa' \mathbin{\hat{+}} (\epsilon = \mathbb{C}) \;?\; 1 \;:\; 0 \end{array}}{\Delta; \Phi; \Gamma \vdash^{\kappa}_{\epsilon} \mathtt{case}_{\mathtt{L}}\ e\ \mathtt{of}\ \mathtt{nil}\ \to\ e_1 \mid \mathtt{cons}(h, tl)\ \to\ e_2 : \tau' \hookrightarrow \begin{array}{l} \mathtt{case}_{\mathtt{L}}\ \ulcorner e \urcorner\ \mathtt{of} \\ \mid\ \mathtt{nil}\ \to\ \ulcorner e_1 \urcorner \\ \mid\ \mathtt{cons}(s, tl)\ \to\ \mathtt{case}(s, h.\ulcorner e_2 \urcorner_{l}, h.\ulcorner e_2 \urcorner_{r}) \end{array}} \;\textbf{caseL}$$

Figure 28: Translation rules, part 3

$$\begin{array}{lcl}
[\![n]\!]_\varphi & = & n \\
[\![x]\!]_\varphi & = & \varphi x \\
[\![0]\!]_\varphi & = & 0 \\
[\![I \mathbin{\hat{+}} 1]\!]_\varphi & = & [\![I]\!]_\varphi + 1 \\
[\![I_1 \mathbin{\hat{+}} I_2]\!]_\varphi & = & [\![I_1]\!]_\varphi + [\![I_2]\!]_\varphi \\
[\![I_1 \mathbin{\hat{-}} I_2]\!]_\varphi & = & [\![I_1]\!]_\varphi - [\![I_2]\!]_\varphi \\
[\![I_1 \mathbin{\hat{\cdot}} I_2]\!]_\varphi & = & [\![I_1]\!]_\varphi \cdot [\![I_2]\!]_\varphi \\
[\![\hat{\frac{I_1}{I_2}}]\!]_\varphi & = & \frac{[\![I_1]\!]_\varphi}{[\![I_2]\!]_\varphi} \\
[\![{I_1}^{I_2}]\!]_\varphi & = & [\![I_1]\!]_\varphi^{[\![I_2]\!]_\varphi} \\
[\![\hat{\lceil I \rceil}]\!]_\varphi & = & \lceil [\![I]\!]_\varphi \rceil \\
[\![\hat{\lfloor I \rfloor}]\!]_\varphi & = & \lfloor [\![I]\!]_\varphi \rfloor \\
[\![\hat{\texttt{log}}_2(I)]\!]_\varphi & = & \texttt{log}_2([\![I]\!]_\varphi) \\
[\![\hat{\texttt{min}}(I_1, I_2)]\!]_\varphi & = & \texttt{min}[\![I_1]\!]_\varphi[\![I_2]\!]_\varphi \\
[\![\hat{\texttt{max}}(I_1, I_2)]\!]_\varphi & = & \texttt{max}[\![I_1]\!]_\varphi[\![I_2]\!]_\varphi \\
[\![\hat{\sum}_{i=I_1}^{I_n} I]\!]_\varphi & = & \sum_{i=[\![I_1]\!]_\varphi}^{[\![I_2]\!]_\varphi} [\![I]\!]_{\varphi \backslash i}
\end{array}$$

Definition 2 (Constraint interpretation)
Let $\varphi \in \mathcal{D}[\![\Delta]\!]$ and $\Delta; \Phi \vdash C$ $\texttt{wf}$. The constraint C is interpreted as follows

$$\begin{array}{lcl}
[\![I_1 \doteq I_2]\!]_\varphi & = & [\![I_1]\!]_\varphi = [\![I_2]\!]_\varphi \\
[\![I_1 \mathbin{\dot{<}} I_2]\!]_\varphi & = & [\![I_1]\!]_\varphi < [\![I_2]\!]_\varphi \\
[\![\dot{\neg} I]\!]_\varphi & = & \neg[\![I]\!]_\varphi \\
[\![\dot{\bot}]\!]_\varphi & = & \bot \\
[\![\dot{\top}]\!]_\varphi & = & \top \\
[\![C_1 \mathbin{\dot{\wedge}} C_2]\!]_\varphi & = & [\![C_1]\!]_\varphi \wedge [\![C_2]\!]_\varphi \\
[\![C_1 \mathbin{\dot{\vee}} C_2]\!]_\varphi & = & [\![C_1]\!]_\varphi \vee [\![C_2]\!]_\varphi
\end{array}$$

Lemma 6 (Index term interpretation soundness)
Let $\varphi \in \mathcal{D}[\![\Delta]\!]$ and $\Delta; \Phi \vdash i : S$. Then

- $S = \dot{\mathbb{N}}$ iff $[\![i]\!]_\varphi \in \mathbb{N}$
- $S = \dot{\mathbb{R}}^+$ iff $[\![i]\!]_\varphi \in \mathbb{R}^+$

Proof. The result follows easily by induction on the sorting derivation. □

Definition 3 (Constraint satisfaction)
Let $\Delta; \Phi \vdash C$ $\texttt{wf}$. Then $\Delta; \Phi \models C$ iff for all $\varphi \in \mathcal{D}[\![\Delta]\!]$, if $[\![\Phi]\!]_\varphi$ holds then $[\![C]\!]_\varphi$ holds.

Appendix B Proofs

Lemma 7 (Evaluation invariants)
If

$$e,\ \sigma \Downarrow^L v,\ \sigma',\ D,\ j$$

then the following hold:

1. $\sigma' \sqsupseteq^L \sigma$

Proof. By induction on the evaluation derivation. □

Lemma 8 (Evaluation is deterministic)
If

$$e,\ \sigma_i \Downarrow^{L} v,\ \sigma_f,\ \varnothing,\ j$$

and

$$e,\ \sigma_i \Downarrow^{L} v',\ \sigma'_f,\ \varnothing,\ j'$$

then $j = j'$, $v' = v$ and $\sigma'_f = \sigma_f$

Proof. By induction on the evaluation derivation. □

Lemma 9 (Graph locations)
If $e,\ \sigma_i \Downarrow^{L} v,\ \sigma_f,\ D,\ j$ and $\mathtt{locs}(e) = \emptyset$ and $\mathtt{cl}(\sigma)$ then $\mathtt{locs}(D) \subseteq \sigma_f \setminus \sigma_i$

Proof. By induction on the evaluation derivation. □

Lemma 10 (Change propagation invariants)
If

$$D,\ \sigma_c,\ \sigma_f,\ \beta \rightsquigarrow \sigma'_f,\ \beta',\ c$$

then the following hold:

1. $\sigma'_f \sqsupseteq^{L_2} \sigma_c$
2. $\mathtt{dom}(\beta') \geq \mathtt{dom}(\beta)$

Proof. The result follows by induction on the length of the queue. □

Lemma 11 (Change propagation under store extension)
If

$$D,\ \sigma_c,\ \sigma_f,\ \beta \rightsquigarrow \sigma'_f,\ \beta',\ c$$

and then for all $\sigma''_f \sqsupseteq^{L_1} \sigma_f$

$$D,\ \sigma_c,\ \sigma''_f,\ \beta \rightsquigarrow \sigma'_f,\ \beta',\ c$$

Proof. The result follows by induction on the length of the queue. □

Lemma 12 (Change propagation under bijection extension)
If

$$D,\ \sigma_c,\ \sigma_f,\ \beta \rightsquigarrow \sigma'_f,\ \beta',\ c$$

then for all β_j such that $\beta \otimes \beta_j$ and $\beta' \otimes \beta_j$ are defined, and $\mathtt{locs}(D) \cap \mathtt{dom}(\beta_j) = \emptyset$ then

$$D,\ \sigma_c,\ \sigma''_f,\ \beta \otimes \beta_j \rightsquigarrow \sigma'_f,\ \beta' \otimes \beta_j,\ c$$

Proof. The result follows by induction on the length of the queue. □

Lemma 13 (Change propagation composition)
If

$$D_1,\ \sigma_c,\ \sigma_f,\ \beta \rightsquigarrow \sigma'_f,\ \beta',\ c_1$$

and

$$D_2,\ \sigma'_f,\ \sigma_f,\ \beta' \rightsquigarrow \sigma''_f,\ \beta'',\ c_2$$

then

$$D_1 + D_2,\ \sigma_c,\ \sigma_f,\ \beta \rightsquigarrow \sigma''_f,\ \beta'',\ c_1 + c_2$$

Proof. The result follows by induction on the length of the first queue. □

Lemma 14 (No free locations)
Assume that

$$\Delta; \Phi; \Gamma \vdash^{\kappa}_{\epsilon} e : \tau \hookrightarrow \ulcorner e \urcorner$$

Then $\texttt{locs}(\ulcorner e \urcorner) = \emptyset$.

Proof. By induction on the derivation. □

Lemma 15 (World extension closure)
The following hold:

1. If $(v_s,\ v_t,\ W) \in \mathcal{V}(\!|\tau|\!)_L$ and $W' \geq W$, then $(v_s,\ v_t,\ W') \in \mathcal{V}(\!|\tau|\!)_L$.
2. If $(v_i,\ v_c,\ v_t,\ W) \in \mathcal{V}[\![\tau]\!]$ and $W' \geq W$, then $(v_i,\ v_c,\ v_t,\ W') \in \mathcal{V}[\![\tau]\!]$.
3. If $(e_s,\ e_t,\ W) \in \mathcal{E}(\!|\tau|\!)^{\kappa}_L$ and $W' \geq W$, then $(e_s,\ e_t,\ W') \in \mathcal{E}(\!|\tau|\!)^{\kappa}_L$.
4. If $(e_i,\ e_c,\ e_t,\ W) \in \mathcal{E}[\![\tau]\!]^{\kappa}$ and $W' \geq W$, then $(e_i,\ e_c,\ e_t,\ W') \in \mathcal{E}[\![\tau]\!]^{\kappa}$.
5. If $(\theta_s,\ \theta_t,\ W) \in \mathcal{G}(\!|\Gamma|\!)_L$ and $W' \geq W$, then $(\theta_s,\ \theta_t,\ W') \in \mathcal{G}(\!|\Gamma|\!)_L$.
6. If $(\theta_i,\ \theta_c,\ \theta_t,\ W) \in \mathcal{G}[\![\Gamma]\!]$ and $W' \geq W$, then $(\theta_i,\ \theta_c,\ \theta_t,\ W') \in \mathcal{G}[\![\Gamma]\!]$.

Proof. First we prove statements (1) and (3) simultaneously, by induction on the type. Then we prove statement (5) by induction on the length of the environment, using statement (1).

Using the statement (1), we can prove statements (2) and (4) simultaneously, by induction on the type. Then we prove statement (6) by induction on the length of the environment, using statement (2). □

Lemma 16 (Value relation projection)
Then the following hold:

1. If $(v_i,\ v_c,\ v_t,\ (\sigma_i,\ \sigma_c,\ \beta,\ j)) \in \mathcal{V}[\![\tau]\!]$ then $(v_i,\ v_t,\ (\sigma_i,\ j)) \in \mathcal{V}(\!|\tau|\!)_{L_1}$ and $\forall\, m,\ (v_c,\ v_t,\ (\sigma_c,\ m)) \in \mathcal{V}(\!|\tau|\!)_{L_2}$
2. If $(\theta_i,\ \theta_c,\ \theta_t,\ (\sigma_i,\ \sigma_c,\ \beta,\ m)) \in \mathcal{G}[\![\Gamma]\!]$ then $(\theta_i,\ \theta_t,\ (\sigma_i,\ j)) \in \mathcal{G}(\!|\Gamma|\!)_{L_1}$ and $\forall\, m,\ (\theta_c,\ \theta_t,\ (\sigma_c,\ m)) \in \mathcal{G}(\!|\Gamma|\!)_{L_2}$

Proof. (1) follows by induction on the type τ. (2) follows by induction in the size of Γ and using (1). □

Lemma 17 (Value relation composition)
Then the following hold:

1. If $(v_i,\ v_t,\ (\sigma_i,\ j)) \in \mathcal{V}(\!|\tau|\!)_{L_1}, \forall\, m,\ (v_c,\ v_t,\ (\sigma_c,\ m)) \in \mathcal{V}(\!|\tau|\!)_{L_2}$ and $\mathbb{C} \trianglelefteq \tau$ then $(\theta_i,\ \theta_c,\ \theta_t,\ (\sigma_i,\ \sigma_c,\ \beta,\ m)) \in \mathcal{G}[\![\Gamma]\!]$
2. If $(\theta_i,\ \theta_t,\ (\sigma_i,\ j)) \in \mathcal{G}(\!|\Gamma|\!)_{L_1}, \forall\, m,\ (\theta_c,\ \theta_t,\ (\sigma_c,\ m)) \in \mathcal{G}(\!|\Gamma|\!)_{L_2}$ and $\mathbb{C} \trianglelefteq \tau$ then $(\theta_i,\ \theta_c,\ \theta_t,\ (\sigma_i,\ \sigma_c,\ \beta,\ m)) \in \mathcal{G}[\![\Gamma]\!]$

Proof. (1) follows by induction on the type τ. (2) follows by induction in the size of Γ and using (1). □

Lemma 18
Assume that:

$$\forall\, m,\ (e_i,\ e_t,\ (\sigma,\ m)) \in \mathcal{E}(\!|\tau|\!)^{\kappa}_L \tag{A}$$

Then, there exist $v_i,\ j,\ v_t,\ \sigma'$:

1. $e_i \Downarrow v_i,\ j$
2. $e_c,\ \sigma \Downarrow^L v_t,\ \sigma',\ \varnothing,\ c$
3. $c \leq \kappa$

4. $\forall\ m,\ (v_i,\ v_t,\ (\sigma',\ m)) \in \mathcal{V}\llparenthesis \tau \rrparenthesis_L$

Proof. We instantiate eq. (A) with step index 1 and obtain v_i, j such that

$$e_i \Downarrow v_i,\ j$$

We can now instantiate eq. (A) with step index $j+1$ and by using the fact that the evaluation relation is deterministic in the source and $j < j+1$ we can obtain v_t, σ_f and c such that

$$e_c,\ \sigma \Downarrow^L v_t,\ \sigma_f,\ \varnothing,\ c$$

and

$$c \leq \kappa$$

To show the third goal we pick an arbitrary m and we instantiate eq. (A) with $m+j+1$. Using the fact that the evaluation relation is deterministic in the source and $j < m+j+1$ we can obtain v'_t and σ'' such that

$$e_c,\ \sigma \Downarrow^L v'_t,\ \sigma'_f,\ \varnothing,\ c'$$

and

$$(v_i,\ v'_t,\ (\sigma'_f,\ m+1)) \in \mathcal{V}\llparenthesis \tau \rrparenthesis_L$$

Using lemma 15 we can show that

$$(v_i,\ v'_t,\ (\sigma'_f,\ m)) \in \mathcal{V}\llparenthesis \tau \rrparenthesis_L$$

Using lemma lemma 8 we can we can show that $v'_t = v_t$ and $\sigma'_f = \sigma_f$, thus

$$(v_i,\ v_t,\ (\sigma_f,\ m)) \in \mathcal{V}\llparenthesis \tau \rrparenthesis_L$$

□

Lemma 19 (Map - Unary interpretation)
Assume that $\varphi \in \mathcal{D}\llbracket\Delta\rrbracket$, $\models \varphi\Phi$ and the following hold

$$(lst_s,\ lst_t,\ (\sigma_i,\ k)) \in \mathcal{V}\llparenthesis \texttt{list}\,[\phi n]^{\phi\alpha}\ \phi\tau \rrparenthesis_L \tag{1}$$

for all $(v_s,\ v_t,\ (\sigma_i,\ m)) \in \mathcal{V}\llparenthesis \phi\tau^{\downarrow\square} \rrparenthesis_L$,

$$(\texttt{inl}\ v_s,\ g\ (\texttt{inl}\ v_t),\ (\sigma_i,\ m)) \in \mathcal{E}\llparenthesis (\phi\tau'^{\downarrow\square} + \phi\tau')^{\mathbb{S}} \rrparenthesis_L^{\phi\kappa} \tag{2}$$

and for all $(v_s,\ v_t,\ (\sigma_i,\ m)) \in \mathcal{V}\llparenthesis \phi\tau \rrparenthesis_L$,

$$(\texttt{inr}\ v_s,\ g\ (\texttt{inr}\ v_t),\ (\sigma_i,\ m)) \in \mathcal{E}\llparenthesis (\phi\tau'^{\downarrow\square} + \phi\tau')^{\mathbb{S}} \rrparenthesis_L^{\phi\kappa'} \tag{3}$$

then

$$(lst_s,\ \texttt{map}\ g\ lst_t,\ (\sigma_i,\ k)) \in \mathcal{E}\llparenthesis \texttt{list}\,[\phi n]^{\phi\alpha}\ \phi\tau' \rrparenthesis_L^{\phi(\alpha\ \hat{\cdot}\ \kappa'\ \hat{+}\ (n\ \hat{-}\ a)\ \hat{\cdot}\ \kappa)}$$

Proof. The proof proceed by induction on ϕn. □

Lemma 20 (Map - Binary interpretation)
Assume that $\varphi \in \mathcal{D}\llbracket\Delta\rrbracket$, $\models \varphi\Phi$ and the following hold

$$(lst_i,\ lst_c,\ lst_t,\ (\sigma_i,\ \sigma_c,\ \beta,\ k)) \in \mathcal{V}\llbracket \texttt{list}\,[\phi n]^{\phi\alpha}\ \phi\tau \rrbracket \tag{1}$$

for all $(v_i,\ v_c,\ v_t,\ (\sigma_i,\ \sigma_c,\ \beta,\ m)) \in \mathcal{V}\llbracket \phi\tau^{\downarrow\square} \rrbracket$,

$$(\texttt{inl}\ v_i,\ \texttt{inl}\ v_c,\ g\ (\texttt{inl}\ v_t),\ (\sigma_i,\ \sigma_c,\ \beta,\ m)) \in \mathcal{E}\llbracket (\phi\tau'^{\downarrow\square} + \phi\tau')^{\mathbb{S}} \rrbracket^{0} \tag{2}$$

and for all $(v_i,\ v_c,\ v_t,\ (\sigma_i,\ \sigma_c,\ \beta,\ m)) \in \mathcal{V}\llbracket \phi\tau \rrbracket$,

$$(\texttt{inr}\ v_i,\ \texttt{inr}\ v_c,\ g\ (\texttt{inr}\ v_t),\ (\sigma_i,\ \sigma_c,\ \beta,\ m)) \in \mathcal{E}\llbracket (\phi\tau'^{\downarrow\square} + \phi\tau')^{\mathbb{S}} \rrbracket^{\kappa} \tag{3}$$

then

$$(lst_i,\ lst_c,\ \texttt{map}\ g\ lst_t,\ (\sigma_i,\ \sigma_c,\ \beta,\ k)) \in \mathcal{E}\llbracket \texttt{list}\,[\phi n]^{\phi\alpha}\ \phi\tau' \rrbracket^{\phi\alpha\ \hat{\cdot}\ \phi\kappa}$$

Proof. We proceed by induction on ϕn. The interesting case is the inductive one. Assume that $\phi n = n' + 1$ for an integer n'. There are two possible cases

- $lst_i = v_i :: vs_i$, $lst_c = v_c :: vs_c$, $lst_t = \texttt{inl}\ v_t :: vs_t$
  From the definition of the logical relation we derive
  $$(v_i,\ v_c,\ v_t,\ (\sigma_i,\ \sigma_c,\ \beta,\ k)) \in \mathcal{V}\llbracket \phi\tau^{\downarrow\Box} \rrbracket \tag{A}$$
  and
  $$(vs_i,\ vs_c,\ vs_t,\ (\sigma_i,\ \sigma_c,\ \beta,\ k)) \in \mathcal{V}\llbracket \texttt{list}\,[\phi n - 1]^{\phi\alpha}\ \phi\tau \rrbracket \tag{B}$$
  From the induction hypothesis applied on eq. (B) we derive
  $$(vs_i,\ vs_c,\ \texttt{map}\ g\ vs_t,\ (\sigma_i,\ \sigma_c,\ \beta,\ k)) \in \mathcal{E}\llbracket \texttt{list}\,[\phi n - 1]^{\phi\alpha}\ \phi\tau' \rrbracket^{\phi\alpha\ \hat{\cdot}\ \phi\kappa} \tag{C}$$
  We instantiate eq. (2) with eq. (A) and we derive
  $$(\texttt{inl}\ v_i,\ \texttt{inl}\ v_c,\ g\ (\texttt{inl}\ v_t),\ (\sigma_i,\ \sigma_c,\ \beta,\ m)) \in \mathcal{E}\llbracket (\phi\tau'^{\downarrow\Box} + \phi\tau')^{\mathbb{S}} \rrbracket^{0} \tag{D}$$
  We can now combine the two statements above, instantiate them with the appropriate stores and bijections and use the definition of `map` in order to obtain
  $$(v_i :: vs_i,\ v_c :: vs_c,\ \texttt{map}\ g\ \texttt{inl}\ v_t :: vs_t,\ (\sigma_i,\ \sigma_c,\ \beta,\ k)) \in \mathcal{E}\llbracket \texttt{list}\,[\phi n]^{\phi\alpha}\ \phi\tau' \rrbracket^{\phi\alpha\ \hat{\cdot}\ \phi\kappa}$$

- $lst_i = v_i :: vs_i$, $lst_c = v_c :: vs_c$, $lst_t = \texttt{inr}\ v_t :: vs_t$
  From the definition of the logical relation we derive
  $$(v_i,\ v_c,\ v_t,\ (\sigma_i,\ \sigma_c,\ \beta,\ k)) \in \mathcal{V}\llbracket \phi\tau \rrbracket \tag{A}$$
  and
  $$(vs_i,\ vs_c,\ vs_t,\ (\sigma_i,\ \sigma_c,\ \beta,\ k)) \in \mathcal{V}\llbracket \texttt{list}\,[\phi n - 1]^{\phi\alpha - 1}\ \phi\tau \rrbracket \tag{B}$$
  From the induction hypothesis applied on eq. (B) we derive
  $$(vs_i,\ vs_c,\ \texttt{map}\ g\ vs_t,\ (\sigma_i,\ \sigma_c,\ \beta,\ k)) \in \mathcal{E}\llbracket \texttt{list}\,[\phi n - 1]^{\phi\alpha - 1}\ \phi\tau' \rrbracket^{\phi\alpha - 1\ \hat{\cdot}\ \phi\kappa} \tag{C}$$
  We instantiate eq. (3) with eq. (A) and we derive
  $$(\texttt{inr}\ v_i,\ \texttt{inr}\ v_c,\ g\ (\texttt{inr}\ v_t),\ (\sigma_i,\ \sigma_c,\ \beta,\ m)) \in \mathcal{E}\llbracket (\phi\tau'^{\downarrow\Box} + \phi\tau')^{\mathbb{S}} \rrbracket^{\kappa} \tag{D}$$
  We can now combine the two statements above, instantiate them with the appropriate stores and bijections and use the definition of `map` in order to obtain
  $$(v_i :: vs_i,\ v_c :: vs_c,\ \texttt{map}\ g\ \texttt{inl}\ v_t :: vs_t,\ (\sigma_i,\ \sigma_c,\ \beta,\ k)) \in \mathcal{E}\llbracket \texttt{list}\,[\phi n]^{\phi\alpha}\ \phi\tau' \rrbracket^{\phi\alpha\ \hat{\cdot}\ \phi\kappa}$$

□

Lemma 21 (Location Flattening)
Let $\Delta;\ \Phi \vdash \tau\ \texttt{wf}$, $\varphi \in \mathcal{D}\llbracket\Delta\rrbracket$ and $\models \varphi\Phi$. Assume that the following hold:

$$e_1,\ \sigma_i \Downarrow^{L_1} v_1,\ \sigma_{f_1},\ \varnothing,\ c_1 \tag{A}$$
$$e_2,\ \sigma_c + \sigma_i \Downarrow^{L_2} v_2,\ \sigma_{f_2} + \sigma_i,\ \varnothing,\ c_2 \tag{B}$$
$$(v_i,\ v_1,\ (\sigma_{f1},\ k)) \in \mathcal{V}\llparenthesis \phi\tau \rrparenthesis_{L_1} \tag{C}$$
$$\forall\ m,\ (v_c,\ v_2,\ (\sigma_{f2} + \sigma_i,\ m)) \in \mathcal{V}\llparenthesis \phi\tau \rrparenthesis_{L_2} \tag{D}$$
$$under\mathbb{C}\tau$$

Then, for all lists of locations $\vec{l_{acc}}$ and $\vec{l'_{acc}}$ and bijections β such that $\texttt{dom}(\beta) \subset \sigma_{f1}$ and $\texttt{dom}'(\beta) \subset \sigma_{f1}$, there exist v, n, $\vec{v^i} = [v^i_1, \ldots, v^i_n]$, $\vec{v^c} = [v^c_1, \ldots, v^c_n]$ and $\vec{l} = [l_1, \ldots, l_n]$, $\vec{l'} = [l'_1, \ldots, l'_n]$ both with all the elements pairwise distinct and furthermore for all $i \in [1, n]$, $l_i \in L_1$, $l_i \notin \texttt{dom}(\sigma_{f1})$, $l'_i \in L_2$ and $l'_i \notin \texttt{dom}(\sigma_{f2})$, such that:

1. $\underline{\mathbf{deep_{aux}}}(e_1,\ \tau,\ \vec{l_{acc}}),\ \sigma_i \Downarrow^{L_1} (v, \vec{l}@\vec{l_{acc}}),\ \sigma_{f1}[\langle \vec{l} \mapsto \vec{v^i} \rangle],\ \varnothing,\ c_1'$
2. $\Delta; \Phi \models c_1' \dot{\leq} c_1 + \texttt{cost}_{\texttt{deep}}(\tau)$
3. $\underline{\mathbf{deep'_{aux}}}(e_2,\ \tau,\ \vec{l'_{acc}}),\ \sigma_c \Downarrow^{L_2} \vec{l'}@\vec{l'_{acc}},\ \sigma_{f2}[\langle \vec{l'} \mapsto \vec{v^c} \rangle],\ \varnothing,\ c_2'$
4. $\Delta; \Phi \models c_2' \dot{\leq} c_2 + \texttt{cost}_{\texttt{deepref}}(\tau)$
5. $(v_i,\ v_c,\ v,\ (\sigma_{f1}[\langle \vec{l} \mapsto \vec{v^i} \rangle],\ \sigma_{f2}[\langle \vec{l'} \mapsto \vec{v^c} \rangle],\ \beta \otimes \vec{l} \mapsto \vec{l'},\ k)) \in \mathcal{V}[\![\tau]\!]$

Proof. We proceed by induction on τ. We immediately exclude the $(A)^{\mathbb{S}}$, $(A)^{\square}$ and $\texttt{unit}$ cases because of the $\mathbb{C} \trianglelefteq \tau$ constraint.

- $(A)^{\mathbb{C}}$

In this case,

$$\underline{\mathbf{deep_{aux}}}(e_1,\ (A)^{\mathbb{C}},\ \vec{l_{acc}}) = \texttt{let}\ l\ =\ \texttt{ref}\ !e_1\ \texttt{in}\ (l, l :: \vec{l_{acc}})$$

and

$$\underline{\mathbf{deep'_{aux}}}(e_2,\ (A)^{\mathbb{C}},\ \vec{v_{acc}}) = \texttt{let}\ l\ =\ \texttt{ref}\ !e_2\ \texttt{in}\ l :: acc$$

The goals are proved as follows:

1. Using the evaluation rules and eq. (A) we can easily derive the following judgment:

$$\underline{\mathbf{deep_{aux}}}(e_1,\ (A)^{\mathbb{C}},\ \vec{l_{acc}}),\ \sigma_i \Downarrow^{L_1} (l, [l]@\vec{l_{acc}}),\ \sigma_{f_1}[\langle l \mapsto \sigma_{f1}(v_1) \rangle],\ \varnothing,\ c_1 + 3$$

and $l \in L_1$, $l \notin \texttt{dom}(\sigma_{f1})$

2. We can trivially show that

$$\Delta; \Phi \models c_1 + 3 \dot{\leq} c_1 + 3$$

3. Similarly, using and eq. (B), we can show:

$$e_2,\ \sigma_c \Downarrow^{L_2} [l']@\vec{l'_{acc}},\ \sigma_{f_2}[\langle l' \mapsto \sigma_{f2}(v_2) \rangle],\ \varnothing,\ c_2 + 3$$

and $l' \in L_2$, $l' \notin \texttt{dom}(\sigma_{f2})$

4. We can trivially show that

$$\Delta; \Phi \models c_2 + 3 \dot{\leq} c_2 + 3$$

5. Finally, we need to show that:

$$(v_i,\ v_c,\ l,\ (\sigma_{f_1}[\langle l \mapsto \sigma_{f1}(v_1) \rangle],\ \sigma_{f_2}[\langle l' \mapsto \sigma_{f2}(v_2) \rangle],\ l \mapsto l',\ k)) \in \mathcal{V}[\![(A)^{\mathbb{C}}]\!]$$

From the definition of the logical relation it suffices to show that

$$(v_i,\ \sigma_f(v_1),\ (\sigma_{f_1}[\langle l \mapsto \sigma_{f1}(v_1) \rangle],\ k)) \in \mathcal{V}_A(\!|A|\!)_{L_1}$$

which we get from eq. (A) and **??** and

$$\forall\ m,\ (v_c,\ \sigma_{f2}(v_1),\ (\sigma_{f_2}[\langle l' \mapsto \sigma_{f2}(v_2)\rangle],\ m)) \in \mathcal{V}_A(\!|A|\!)_{L_2}$$

which we get from eq. (B) and the same lemma.

- $\tau_1 \times \tau_2$
  We can derive that $\Delta; \Phi \models \mathbb{C} \trianglelefteq \tau_1$, $\Delta; \Phi \models \mathbb{C} \trianglelefteq \tau_2$. From eq. (C), eq. (D) and the definition of the logical relation we can derive that $v_i = (v_{il}, v_{ir})$, $v_c = (v_{cl}, v_{cr})$, $v_1 = (v_{1l}, v_{1r})$, $v_2 = (v_{2l}, v_{2r})$ and

$$(v_{il},\ v_{1l},\ (\sigma_f,\ k)) \in \mathcal{V}(\!|\tau_1|\!)_{L_1} \tag{C1}$$

$$(v_{ir},\ v_{1r},\ (\sigma_f,\ k)) \in \mathcal{V}(\!|\tau_2|\!)_{L_1} \tag{C2}$$

$$\forall\ m,\ (v_{cl},\ v_{2l},\ (\sigma'_f,\ m)) \in \mathcal{V}(\!|\tau_1|\!)_{L_2} \tag{D1}$$

$$\forall\ m,\ (v_{cr},\ v_{2r},\ (\sigma'_f,\ m)) \in \mathcal{V}(\!|\tau_2|\!)_{L_2} \tag{D2}$$

Furthermore we can easily show that:

$$\texttt{fst}\ (v_{1l}, v_{1r}),\ \sigma_{f1} \Downarrow^{L_1} v_{1l},\ \sigma_{f1},\ \varnothing,\ 1 \tag{A1}$$

and

$$\texttt{fst}\ (v_{2l}, v_{2r}),\ \sigma_{f2} + \sigma_i \Downarrow^{L_2} v_{2l},\ \sigma_{f2} + \sigma_i,\ \varnothing,\ 1 \tag{B1}$$

We can instantiate the induction hypothesis for τ_1 with eqs. (A1) to (D1), $\vec{l_{acc}}$, and $\vec{l'_{acc}}$ and obtain v_l, c_{1l}, c_{2l}, $\vec{v_1^i} = [v_1^i, \ldots, v_{n_1}^i]$, $\vec{v_1^c} = [v_1^c, \ldots, v_{n_1}^c]$ and $\vec{l_1} = [l_1, \ldots, l_{n_1}]$, $\vec{l'_1} = [l'_1, \ldots, l'_{n_1}]$ both with all the elements pairwise distinct and furthermore for all $i \in [1, n_1]$, $l_i \in L_1$, $l_i \notin \texttt{dom}(\sigma_{f1})$, $l'_i \in L_2$ and $l'_i \notin \texttt{dom}(\sigma_{f2})$, such that:

$$\begin{gathered}\underline{\textbf{deep}_{\textbf{aux}}}(\texttt{fst}\ (v_{1l}, v_{1r}),\ \tau_1,\ \vec{l_{acc}}),\ \sigma_{f1} \\ \Downarrow^{L_1} \\ (v_l, \vec{l_1}@\vec{l_{acc}}),\ \sigma_{f1}[\langle \vec{l_1} \mapsto \vec{v_1^i}\rangle],\ \varnothing,\ c_{1l}\end{gathered} \tag{1.1}$$

$$\Delta; \Phi \models c_{1l} \dot{\leq} 1 + \texttt{cost}_{\texttt{deep}}(\tau_1) \tag{1.2}$$

$$\begin{gathered}\underline{\textbf{deep}'_{\textbf{aux}}}(\texttt{fst}\ (v_{2l}, v_{2r}),\ \tau_1,\ \vec{l'_{acc}}),\ \sigma_{f2} + \sigma_i \\ \Downarrow^{L_2} \\ \vec{l'_1}@\vec{l'_{acc}},\ \sigma_{f2}[\langle \vec{l'_1} \mapsto \vec{v_1^c}\rangle] + \sigma_i,\ \varnothing,\ c_{2l}\end{gathered} \tag{1.3}$$

$$\Delta; \Phi \models c_{2l} \dot{\leq} 1 + \texttt{cost}_{\texttt{deep}}(\tau_1) \tag{1.4}$$

and

$$(v_{il},\ v_{cl},\ v_l,\ (\sigma_{f1}[\langle \vec{l_1} \mapsto \vec{v_1^i}\rangle],\ \sigma_{f2}[\langle \vec{l'_1} \mapsto \vec{v_1^c}\rangle],\ \beta \otimes \vec{l_1} \mapsto \vec{l'_1},\ k)) \in \mathcal{V}[\![\tau_1]\!] \tag{1.5}$$

We can also show the following evaluation judgments:

$$\texttt{snd}\ (v_{1l}, v_{1r}),\ \sigma_{f1}[\langle \vec{l_1} \mapsto \vec{v_1^i} \rangle] \Downarrow^{L_1} v_{1r},\ \sigma_{f1}[\langle \vec{l_1} \mapsto \vec{v_1^i} \rangle],\ \varnothing,\ 1 \tag{A2}$$

and

$$\texttt{snd}\ (v_{2l}, v_{2r}),\ \sigma_{f2}[\langle \vec{l'_1} \mapsto \vec{v_1^c} \rangle] + \sigma_i \Downarrow^{L_2} v_{2r},\ \sigma_{f2}[\langle \vec{l'_1} \mapsto \vec{v_1^c} \rangle] + \sigma_i,\ \varnothing,\ 1 \tag{B2}$$

We can instantiate the induction hypothesis for τ_2 with eqs. (A2) to (D2), $\vec{l_1}@\vec{l_{acc}}$, and $\vec{v_1^c}@\vec{v_{acc}}$ and obtain v_r, $\vec{v_2^i} = [v^i_{n_1+1}, \dots, v^i_n]$, $\vec{v_1^c} = [v^c_{n_1+1}, \dots, v^c_n]$ and $\vec{l_2} = [l_{n_1+1}, \dots, l_n]$, $\vec{l'_2} = [l'_{n_1+1}, \dots, l'_n]$ both with all the elements pairwise distinct and furthermore for all $i \in [1, n_1]$, $l_i \in L_1$, $l_i \notin \texttt{dom}(\sigma_{f1}[\langle \vec{l_1} \mapsto \vec{v_1^i} \rangle])$, $l'_i \in L_2$ and $l'_i \notin \texttt{dom}(\sigma_{f2}[\langle \vec{l'_1} \mapsto \vec{v_1^c} \rangle])$, such that

$$\begin{array}{c} \underline{\mathbf{deep}_{\mathbf{aux}}}(\texttt{snd}\ (v_{2l}, v_{2r}),\ \tau_2,\ \vec{l_1}@\vec{l_{acc}}),\ \sigma_{f1}[\langle \vec{l_1} \mapsto \vec{v_1^i} \rangle] + \sigma_i \\ \Downarrow^{L_1} \\ (v_r, \vec{l_2}@\vec{l_1}@\vec{l_{acc}}),\ \sigma_{f1}[\langle \vec{l_1} \mapsto \vec{v_1^i} \rangle][\langle \vec{l_2} \mapsto \vec{v_2^i} \rangle] + \sigma_i,\ \varnothing,\ c_{1r} \end{array} \tag{2.1}$$

$$\Delta; \Phi \models c_{1r} \mathrel{\dot{\leq}} 1 + \texttt{cost}_{\texttt{deep}}(\tau_2) \tag{2.2}$$

$$\begin{array}{c} \underline{\mathbf{deep}'_{\mathbf{aux}}}(\texttt{snd}\ (v_{2l}, v_{2r}),\ \tau_1,\ \vec{l'_1}@\vec{l'_{acc}}),\ \sigma_{f2}[\langle \vec{l'_1} \mapsto \vec{v_1^c} \rangle] \\ \Downarrow^{L_2} \\ \vec{l'_2}@\vec{l'_1}@\vec{l'_{acc}},\ \sigma_{f2}[\langle \vec{l'_1} \mapsto \vec{v_1^c} \rangle][\langle \vec{l'_2} \mapsto \vec{v_2^c} \rangle],\ \varnothing,\ c_{2r} \end{array} \tag{2.3}$$

$$\Delta; \Phi \models c_{2r} \mathrel{\dot{\leq}} 1 + \texttt{cost}_{\texttt{deepref}}(\tau_2) \tag{2.4}$$

$$(v_{ir},\ v_{cr},\ v_r,\ (\sigma_{f1}[\langle \vec{l_1} \mapsto \vec{v_1^i} \rangle][\langle \vec{l_2} \mapsto \vec{v_2^i} \rangle],\ \sigma_{f2}[\langle \vec{l'_1} \mapsto \vec{v_1^c} \rangle][\langle \vec{l'_2} \mapsto \vec{v_2^c} \rangle],\ \beta \otimes \vec{l_1} \mapsto \vec{l'_1} \otimes \vec{l_2} \mapsto \vec{l'_2},\ k)) \in \mathcal{V}[\![\tau_2]\!] \tag{2.5}$$

Let $\vec{l} = \vec{l_2}@\vec{l_1}$, $\vec{l'} = \vec{l'_2}@\vec{l'_1}$, $\vec{v^i} = \vec{v_2^i}@\vec{v_1^i}$ and $\vec{v^c} = \vec{v_2^c}@\vec{v_1^c}$. It is easy to see that all the elements in l (resp l') are pairwise different, drawn from L_1 (resp. L_2), and for all $l \in \vec{l}$, $l \notin \texttt{dom}(\sigma_{f1})$ (resp. for all $l \in \vec{l'}$, $l \notin \texttt{dom}(\sigma_{f2})$). We can now show the goals

1. From eq. (A), eq. (1.1) and eq. (2.1) we can derive that

$$\begin{array}{c} \underline{\mathbf{deep}_{\mathbf{aux}}}(e_1,\ \tau_1 \times \tau_2,\ \vec{l_{acc}}),\ \sigma_i \\ \Downarrow^{L_1} \\ ((v_l, v_r), \vec{l}@\vec{l_{acc}}),\ \sigma_{f1}[\langle \vec{l} \mapsto \vec{v^i} \rangle],\ \varnothing,\ c_1 + c_{1l} + c_{1r} \end{array}$$

2. From eq. (1.2) and eq. (2.2) we can derive that

$$\Delta; \Phi \models c_1 + c_{1l} + c_{1r} + 8 \mathrel{\dot{\leq}} c_1 + \texttt{cost}_{\texttt{deep}}(\tau_1) + \texttt{cost}_{\texttt{deep}}(\tau_2) + 10$$

3. Similarly from eq. (B), eq. (1.3) and eq. (2.3) we can derive that

$$\begin{array}{c} \underline{\mathbf{deep}'_{\mathbf{aux}}}(e_2,\ \tau_1 \times \tau_2,\ \vec{l'_{acc}}),\ \sigma_c \\ \Downarrow^{L_2} \\ \vec{l'}@\vec{l'_{acc}},\ \sigma_{f2}[\langle \vec{l'} \mapsto \vec{v^c} \rangle],\ \varnothing,\ c_2 + c_{2l} + c_{2r} \end{array}$$

4. From eq. (1.4) and eq. (2.4) we can derive that

$$\Delta; \Phi \models c_2 + c_{2l} + c_{2r} + 2 \dot{\leq} c_2 + \texttt{cost}_{\texttt{deep}}(\tau_1) + \texttt{cost}_{\texttt{deep}}(\tau_2) + 4$$

5. Finally, we need to show that

$$((v_{il}, v_{ir}),\ (v_{cl}, v_{cr}),\ (v_l, v_r),\ (\sigma_{f1}[\langle\vec{l} \mapsto \vec{v^i}\rangle],\ \sigma_{f2}[\langle\vec{l'} \mapsto \vec{v^c}\rangle],\ \beta \otimes \vec{l} \mapsto \vec{l'},\ k)) \in \mathcal{V}[\![\tau_1 \times \tau_2]\!]$$

From eq. (1.5) and **??** we can derive that

$$(v_{il},\ v_{cl},\ v_l,\ (\sigma_{f1}[\langle\vec{l} \mapsto \vec{v^i}\rangle],\ \sigma_{f2}[\langle\vec{l'} \mapsto \vec{v^c}\rangle],\ \beta \otimes \vec{l} \mapsto \vec{l'},\ k)) \in \mathcal{V}[\![\tau_1]\!]$$

From eq. (2.5) we can derive that

$$(v_{ir},\ v_{cr},\ v_r,\ (\sigma_{f1}[\langle\vec{l} \mapsto \vec{v^i}\rangle],\ \sigma_{f2}[\langle\vec{l'} \mapsto \vec{v^c}\rangle],\ \beta \otimes \vec{l} \mapsto \vec{l'},\ k)) \in \mathcal{V}[\![\tau_2]\!]$$

From the above and the definition and the logical relation follows the result.

- $\tau = \texttt{list}\,[m]^{\alpha}\ \tau$

We can derive that $\mathbb{C} \trianglelefteq \tau$ and $\Delta; \Phi \models a \doteq m$. We distinguish the following cases.

▶ $\models \phi n \doteq 0$ and $\models \phi a \doteq 0$
Using eq. (C), eq. (D) and the definition of the logical relation we can derive $v_i = nil$, $v_c = nil$, $v_{t1} = []$ and $v_{t2} = []$

▶ $\models 0 \dot{<} \varphi m$ and $\models 0 \dot{<} \varphi a$
Using eq. (C), eq. (D) and the definition of the logical relation we can derive $v_i = v_i' :: vs_i$, $v_c = v_c' :: vs_c$, $v_{t1} = \texttt{inr}\ v_{t1}' :: vs_{t1}$ and $v_{t2} = \texttt{inr}\ v_{t2}' :: vs_{t2}$ for some v_i', vs_i, v_c', vs_c, v_{t1}', vs_{t1}, v_{t2}' and vs_{t2}. We can also derive that

$$(v_i',\ v_{t1}',\ (\sigma_{f1},\ k)) \in \mathcal{V}(\!|\varphi\tau|\!)_{L_1} \tag{C1}$$

$$(vs_i,\ vs_{t1},\ (\sigma_{f1},\ k)) \in \mathcal{V}(\!|\texttt{list}\,[\varphi m \mathbin{\hat{-}} 1]^{\varphi\alpha \mathbin{\hat{-}} 1}\ \varphi\tau|\!)_{L_1} \tag{C2}$$

$$\forall\ m,\ (v_c',\ v_{t2}',\ (\sigma_{f2} + \sigma_i,\ m)) \in \mathcal{V}(\!|\varphi\tau|\!)_{L_2} \tag{D1}$$

$$\forall\ m,\ (vs_c,\ vs_{t2},\ (\sigma_{f2} + \sigma_i,\ m)) \in \mathcal{V}(\!|\texttt{list}\,[\varphi m \mathbin{\hat{-}} 1]^{\varphi\alpha \mathbin{\hat{-}} 1}\ \varphi\tau|\!)_{L_2} \tag{D2}$$

Furthermore we can easily show that

$$vs_{t1},\ \sigma_{f1} \Downarrow^{L_1} vs_{t1},\ \sigma_{f1},\ \varnothing,\ 0 \tag{A1}$$

and

$$vs_{t2},\ \sigma_{f2} + \sigma_i \Downarrow^{L_2} vs_{t2},\ \sigma_{f2} + \sigma_i,\ \varnothing,\ 0 \tag{B1}$$

We can instantiate the induction hypothesis with $\texttt{list}\,[n \mathbin{\hat{-}} 1]^{a \mathbin{\hat{-}} 1}\ \tau$, eqs. (A1) to (D1), $\vec{l_{acc}}$, and $\vec{l'_{acc}}$ and obtain vs_t, $\vec{v^i_1} = [v^i_1, \ldots, v^i_{n_1}]$, $\vec{v^c_1} = [v^c_1, \ldots, v^c_{n_1}]$ and $\vec{l_1} = [l_1, \ldots, l_{n_1}]$, $\vec{l'_1} = [l'_1, \ldots, l'_{n_1}]$ both with all the elements pairwise distinct and furthermore for all $i \in [1, n_1]$, $l_i \in L_1$, $l_i \notin \texttt{dom}(\sigma_{f1})$, $l'_i \in L_2$ and $l'_i \notin \texttt{dom}(\sigma_{f2})$, such that

$$\begin{array}{c}\underline{\mathbf{deep_{aux}}}(vs_{t1},\ \vec{l_{acc}},\ \texttt{list}\,[n \mathbin{\hat{-}} 1]^{a \mathbin{\hat{-}} 1}\ \tau),\ \sigma_{f1} \\ \Downarrow^{L_1} \\ (vs_t, \vec{l_1}@\vec{l_{acc}}),\ \sigma_{f1}[\langle \vec{l_1} \mapsto \vec{v^i_1}\rangle],\ \varnothing,\ c_{tl1}\end{array} \tag{1.1}$$

$$\Delta; \Phi \models c_{tl1} \mathrel{\dot{\leq}} \texttt{cost}_{\texttt{deep}}(\texttt{list}\,[n \mathbin{\hat{-}} 1]^{a \mathbin{\hat{-}} 1}\ \tau) \tag{1.2}$$

$$\begin{array}{c}\underline{\mathbf{deep'_{aux}}}(vs_{t2},\ \vec{l'_{acc}},\ \texttt{list}\,[n \mathbin{\hat{-}} 1]^{a \mathbin{\hat{-}} 1}\ \tau),\ \sigma_{f2} + \sigma_i \\ \Downarrow^{L_2} \\ \vec{l'_1}@\vec{l'_{acc}},\ \sigma_{f2}[\langle \vec{l'_1} \mapsto \vec{v^c_1}\rangle] + \sigma_i,\ \varnothing,\ c_{tl2}\end{array} \tag{1.3}$$

$$\Delta; \Phi \models c_{tl1} \mathrel{\dot{\leq}} \texttt{cost}_{\texttt{deepref}}(\texttt{list}\,[n \mathbin{\hat{-}} 1]^{a \mathbin{\hat{-}} 1}\ \tau) \tag{1.4}$$

and

$$(vs_i,\ vs_c,\ vs_t,\ (\sigma_{f1}[\langle \vec{l_1} \mapsto \vec{v^i_1}\rangle],\ \sigma_{f2}[\langle \vec{l'_1} \mapsto \vec{v^c_1}\rangle],\ \beta \otimes \vec{l_1} \mapsto \vec{l'_1},\ k)) \in \mathcal{V}[\![\texttt{list}\,[\phi n \mathbin{\hat{-}} 1]^{\phi a \mathbin{\hat{-}} 1}\ \phi\tau]\!] \tag{1.5}$$

Using **??** we can derive

$$(v'_i,\ v'_{t1},\ (\sigma_{f1}[\langle \vec{l_1} \mapsto \vec{v^i_1}\rangle],\ k)) \in \mathcal{V}(\!|\varphi\tau|\!)_{L_1} \tag{C3}$$

$$\forall\ m,\ (v'_c,\ v'_{t2},\ (\sigma_{f2}[\langle \vec{l'_1} \mapsto \vec{v^c_1}\rangle] + \sigma_i,\ m)) \in \mathcal{V}(\!|\varphi\tau|\!)_{L_2} \tag{D3}$$

We can also show the following evaluation judgments:

$$v'_{t1},\ \sigma_{f1}[\langle \vec{l_1} \mapsto \vec{v^i_1}\rangle] \Downarrow^{L_1} v'_{t1},\ \sigma_{f1}[\langle \vec{l_1} \mapsto \vec{v^i_1}\rangle],\ \varnothing,\ 0 \tag{A2}$$

and

$$v'_{t2},\ \sigma_{f2}[\langle \vec{l'_1} \mapsto \vec{v^c_1}\rangle] + \sigma_i \Downarrow^{L_2} v'_{t2},\ \sigma_{f2}[\langle \vec{l'_1} \mapsto \vec{v^c_1}\rangle] + \sigma_i,\ \varnothing,\ 0 \tag{B2}$$

We can instantiate the induction hypothesis with τ, eqs. (A2) to (D3), $\vec{l_1}@\vec{l_{acc}}$, and $\vec{l'_1}@\vec{l'_{acc}}$ and obtain v_t, $\vec{v^i_2} = [v^i_{n_1+1}, \ldots, v^i_n]$, $\vec{v^c_1} = [v^c_{n_1+1}, \ldots, v^c_n]$ and $\vec{l_2} = [l_{n_1+1}, \ldots, l_n]$, $\vec{l'_2} = [l'_{n_1+1}, \ldots, l'_n]$ both with all the elements pairwise distinct and furthermore for all $i \in [1, n_1]$, $l_i \in L_1$, $l_i \notin \texttt{dom}(\sigma_{f1}[\langle \vec{l_1} \mapsto \vec{v^i_1}\rangle])$, $l'_i \in L_2$ and $l'_i \notin \texttt{dom}(\sigma_{f2}[\langle \vec{l'_1} \mapsto \vec{v^c_1}\rangle])$, such that

$$\begin{array}{c}\underline{\mathbf{deep_{aux}}}(v'_{t1},\ \vec{l_1}@\vec{l_{acc}},\ \tau),\ \sigma_{f1}[\langle \vec{l_1} \mapsto \vec{v^i_1}\rangle] \\ \Downarrow^{L_1} \\ (v_t, \vec{l_2}@\vec{l_1}@\vec{l_{acc}}),\ \sigma_{f1}[\langle \vec{l_1} \mapsto \vec{v^i_1}\rangle][\langle \vec{l_2} \mapsto \vec{v^i_2}\rangle],\ \varnothing,\ c_{h1}\end{array} \tag{2.1}$$

$$\Delta; \Phi \models c_{h1} \mathrel{\dot{\leq}} \texttt{cost}_{\texttt{deep}}(\tau) \tag{2.2}$$

$$\underline{\mathbf{deep'_{aux}}}(v'_{t2},\ \vec{l'_1}@\vec{l'_{acc}},\ \tau),\ \sigma_{f2}[\langle\vec{l'_1} \mapsto \vec{v^c_1}\rangle] \\ \Downarrow^{L_2} \\ \vec{l'_2}@\vec{l'_1}@\vec{l'_{acc}},\ \sigma_{f2}[\langle\vec{l'_1} \mapsto \vec{v^c_1}\rangle][\langle\vec{l'_2} \mapsto \vec{v^c_2}\rangle],\ \varnothing,\ c_{h2} \tag{2.3}$$

$$\Delta; \Phi \models c_{h1} \mathrel{\dot{\leq}} \mathtt{cost}_{\mathtt{deepref}}(\tau) \tag{2.4}$$

and

$$(v_{i'},\ v'_c,\ v_t,\ (\sigma_{f1}[\langle\vec{l_1} \mapsto \vec{v^i_1}\rangle][\langle\vec{l_2} \mapsto \vec{v^i_2}\rangle],\ \sigma_{f2}[\langle\vec{l'_1} \mapsto \vec{v^c_1}\rangle][\langle\vec{l'_2} \mapsto \vec{v^c_2}\rangle] + \sigma_i,\ \beta \otimes \vec{l_1} \mapsto \vec{l'_1} \otimes \vec{l_2} \mapsto \vec{l'_2},\ k)) \in \mathcal{V}[\![\varphi\tau]\!] \tag{2.5}$$

Let $\vec{l} = \vec{l_2}@\vec{l_1}$, $\vec{l'} = \vec{l'_2}@\vec{l'_1}$, $\vec{v^i} = \vec{v^i_2}@\vec{v^i_1}$ and $\vec{v^c} = \vec{v^c_2}@\vec{v^c_1}$. It is easy to see that all the elements in l (resp l') are pairwise different, drawn from L_1 (resp. L_2), and for all $l \in \vec{l}$, $l \notin \mathtt{dom}(\sigma_{f1})$ (resp. for all $l \in \vec{l'}$, $l \notin \mathtt{dom}(\sigma_{f2})$). Also note that

$$\underline{\mathbf{deep_{aux}}}(vs_{t1},\ \vec{l_{acc}},\ \mathtt{list}\,[n \mathbin{\hat{-}} 1]^{a \mathbin{\hat{-}} 1}\ \tau) = \\ \mathtt{foldr}\ (\lambda h.\, \lambda a.\, \mathtt{case}(h, h_l.\bot, h_r.\mathtt{let}\ p\ =\ \underline{\mathbf{deep_{aux}}}(h_r,\ \tau,\ a)\ \mathtt{in}\ (\mathtt{inr}\ (\mathtt{fst}\ p) :: \mathtt{fst}\ acc, \mathtt{snd}\ p))) \\ vs_{t1}\ ([], \vec{l_{acc}})$$

and

$$\underline{\mathbf{deep'_{aux}}}(vs_{t2},\ \vec{l'_{acc}},\ \mathtt{list}\,[n \mathbin{\hat{-}} 1]^{a \mathbin{\hat{-}} 1}\ \tau) = \\ \mathtt{foldr}\ (\lambda h.\, \lambda a.\, \mathtt{case}(h, h_l.\bot, h_r.\mathtt{inr}\ \underline{\mathbf{deep'_{aux}}}(h_r,\ \tau,\ a)))\ vs_{t2}\ \vec{l'_{acc}}$$

We can now show the goals.

1. Using eq. (1.1) and eq. (2.1) we can show that

$$\underline{\mathbf{deep_{aux}}}(e_1,\ \vec{l_{acc}},\ \mathtt{list}\,[n]^a\ \tau),\ \sigma_i \\ \Downarrow^{L_1} \\ (\mathtt{inr}\ v_t :: vs_t, \vec{l_2}@\vec{l_1}@\vec{l_{acc}}),\ \sigma_{f1}[\langle\vec{l_1} \mapsto \vec{v^i_1}\rangle][\langle\vec{l_2} \mapsto \vec{v^i_2}\rangle],\ \varnothing,\ c_1 + c_{h1} + c_{tl1} + 10$$

2. Using eq. (1.2) and eq. (2.2) we can show that

$$\Delta; \Phi \models c_1 + c_{h1} + c_{tl1} + 10 \mathrel{\dot{\leq}} c_1 + \mathtt{cost}_{\mathtt{deep}}(\tau) + \mathtt{cost}_{\mathtt{deep}}(\mathtt{list}\,[n \mathbin{\hat{-}} 1]^{a \mathbin{\hat{-}} 1}\ \tau) + 10$$

3. Using eq. (1.3) and eq. (2.3) we can show that

$$\underline{\mathbf{deep'_{aux}}}(e_2,\ \vec{l'_{acc}},\ \mathtt{list}\,[n]^a\ \tau),\ \sigma_c \\ \Downarrow^{L_2} \\ \vec{l'_2}@\vec{l'_1}@\vec{l'_{acc}},\ \sigma_{f2}[\langle\vec{l'_1} \mapsto \vec{v^c_1}\rangle][\langle\vec{l'_2} \mapsto \vec{v^c_2}\rangle],\ \varnothing,\ c_1 + c_{h2} + c_{tl2} + 6$$

4. Using eq. (1.4) and eq. (2.4) we can show that

$$\Delta; \Phi \models c_1 + c_{h2} + c_{tl2} + 6 \dot{\leq} c_2 + \texttt{cost}_{\texttt{deepref}}(\tau) + \texttt{cost}_{\texttt{deepref}}(\texttt{list}\ [n \hat{-} 1]^{a \hat{-} 1}\ \tau) + 6$$

5. Using eq. (1.5) and eq. (2.5) we can show that

$$(v_i' :: vs_i,\ v_c' :: vs_c,\ \texttt{inr}\, v_t :: vs_t,\ (\sigma_{f1}[\langle \vec{l_1} \mapsto \vec{v_1^i} \rangle][\langle \vec{l_2} \mapsto \vec{v_2^i} \rangle],\ \sigma_{f2}[\langle \vec{l_1'} \mapsto \vec{v_1^c} \rangle][\langle \vec{l_2'} \mapsto \vec{v_2^c} \rangle] {+} \sigma_i,\ \beta \otimes \vec{l_1} \mapsto \vec{l_1'} \otimes \vec{l_2} \mapsto \vec{l_2'}, \tag{2.3}$$

□

Corollary 22 (Location Flattening)
Assume that the following hold:

$$e_1,\ \sigma_i \Downarrow^{L_1} v_1,\ \sigma_{f_1},\ \varnothing,\ c_1 \tag{A}$$

$$e_2,\ \sigma_c \Downarrow^{L_2} v_2,\ \sigma_{f_2},\ \varnothing,\ c_2 \tag{B}$$

$$(v_i,\ v_1,\ (\sigma_{f1},\ k)) \in \mathcal{V}(\!|\tau|\!)_{L_1} \tag{C}$$

$$\forall\ m,\ (v_c,\ v_2,\ (\sigma_{f2},\ m)) \in \mathcal{V}(\!|\tau|\!)_{L_2} \tag{D}$$

$$\mathbb{C} \trianglelefteq \tau$$

$$|\tau| = n$$

Then there exist v, $\vec{v^i} = [v_1^i, \ldots, v_n^i]$, $\vec{v^c} = [v_1^c, \ldots, v_n^c]$ and $\vec{l} = [l_1, \ldots, l_n]$, $\vec{l'} = [l_1', \ldots, l_n']$ both with all the elements pairwise distinct and furthermore for all $i \in [1, n]$, $l_i \in L_1$, $l_i \notin \texttt{dom}(\sigma_{f1})$, $l_i' \in L_2$ and $l_i' \notin \texttt{dom}(\sigma_{f2})$, such that:

1. $\underline{\textbf{deep}}(e_1,\ \tau),\ \sigma_i \Downarrow^{L_1} (v, \vec{l}),\ \sigma_{f1}[\langle \vec{l} \mapsto \vec{v^i} \rangle],\ \varnothing,\ c_1 + \texttt{cost}_{\texttt{deep}}(\tau)$
2. $\underline{\textbf{deep}'}(e_2,\ \tau),\ \sigma_c \Downarrow^{L_2} \vec{l'},\ \sigma_{f2}[\langle \vec{l'} \mapsto \vec{v^c} \rangle],\ \varnothing,\ c_2 + \texttt{cost}_{\texttt{deepref}}(\tau)$
3. $(v_i,\ v_c,\ v,\ (\sigma_{f1}[\langle \vec{l} \mapsto \vec{v^i} \rangle],\ \sigma_{f2}[\langle \vec{l'} \mapsto \vec{v^c} \rangle],\ k,\ \vec{l} \mapsto \vec{l'})) \in \mathcal{V}[\![\tau]\!]$

Proof. It follows immediately from lemma 21 after unfolding the definitions of $\underline{\textbf{deep}}(e_1,\ \tau)$ and $\underline{\textbf{deep}'}(e_2,\ \tau)$. □

Lemma 23 (Location Dereferencing)
Assume that the following hold

$$e_t,\ \sigma_i \Downarrow^{L_1} v_t,\ \sigma_f,\ \varnothing,\ c \tag{A}$$

$$(v_i,\ v_c,\ v_t,\ (\sigma_f,\ \emptyset,\ \emptyset,\ m)) \in \mathcal{V}[\![\tau]\!] \tag{B}$$

Then for all β and σ_c,

1. $\underline{\textbf{conv}}(e_t,\ \tau,\ \tau^{\downarrow\square}),\ \sigma_i \Downarrow^{L_1} v_t',\ \sigma_i,\ \varnothing,\ c + \texttt{cost}_{\texttt{conv}}(\tau, \tau^{\downarrow\square})$
2. $(v_i,\ v_c,\ ,\ (\sigma_i,\ \sigma_c,\ \beta,\ m)) \in \mathcal{V}[\![\tau^{\downarrow\square}]\!]$

Proof. We proceed by induction on τ. Some of the cases are shown below.

- **$\tau = (A)^\mu$ when $\mu = \square$ or $\mu = \mathbb{S}$**

In this case $\underline{\textbf{conv}}(e,\ \tau,\ \tau^{\downarrow\square}) = e$ and the result follows immediately from the premises and the definition of $\mathcal{V}[\![\cdot]\!]$.

- $\tau = (A)^{\mathbb{C}}$
  In this case $\underline{\textbf{conv}}(e,\ (A)^{\mathbb{C}},\ (A)^{\square}) = !e$. The goals are proved as follows:

  1. Using eq. (A), we can easily show that

  $$!e,\ \sigma_i \Downarrow^{L_1} \sigma_f(v),\ \sigma_f,\ \varnothing,\ c+1$$

  2. We pick an arbitrary β. We need to show that

  $$(v_i,\ v_c,\ \sigma_f(v),\ (\sigma_f,\ \sigma_c,\ \beta,\ m)) \in \mathcal{V}[\![(A)^{\square}]\!]$$

  It suffices to show that

  $$(v_i,\ v_c,\ \sigma_f(v),\ (\sigma_f,\ \sigma_c,\ \emptyset,\ m)) \in \mathcal{V}_A[\![A]\!]$$

  Which we get by unfolding eq. (B).

- $\tau = \tau_1 \times \tau_2$
  By unfolding eq. (B) we can show that $v_i = (v_{il}, v_{ir})$, $v_c = (v_{cl}, v_{cr})$ and $v = (v_l, v_r)$ for some v_{il}, v_{ir},, v_{cl}, v_{cr}, v_l, and v_r, and furthermore that

$$(v_{il},\ v_{cl},\ v_l,\ (\sigma_f,\ \sigma_c,\ \emptyset,\ m)) \in \mathcal{V}[\![\tau_1]\!] \tag{B1}$$

and

$$(v_{ir},\ v_{cr},\ v_r,\ (\sigma_f,\ \sigma_c,\ \emptyset,\ m)) \in \mathcal{V}[\![\tau_2]\!] \tag{B2}$$

We can also show that

$$\texttt{fst}\ (v_l, v_r),\ \sigma_f \Downarrow^{L_1} v_l,\ \sigma_f,\ \varnothing,\ 1 \tag{A1}$$

$$\texttt{snd}\ (v_l, v_r),\ \sigma_f \Downarrow^{L_1} v_r,\ \sigma_f,\ \varnothing,\ 1 \tag{A2}$$

We can now instantiate the inductive hypothesis for τ_1 with eq. (A1) and eq. (B1) and obtain v_l' such that

$$\underline{\textbf{conv}}(\texttt{fst}\ (v_l, v_r),\ \tau_1,\ \tau_1^{\downarrow\square}),\ \sigma_f \Downarrow^{L_1} v_l',\ \sigma_f,\ \varnothing,\ 1 + \texttt{cost}_{\texttt{conv}}(\tau_1, \tau_1^{\downarrow\square}) \tag{C1}$$

and

$$\forall\beta,\ (v_{il},\ v_{cl},\ v_l',\ (\sigma_f,\ \sigma_c,\ \emptyset,\ m)) \in \mathcal{V}[\![\tau_1^{\downarrow\square}]\!] \tag{C2}$$

Similarly, we instantiate the inductive hypothesis for τ_2 with eq. (A2) and eq. (B2) and obtain v_r' such that

$$\forall\beta,\ \underline{\textbf{conv}}(\texttt{snd}\ (v_l, v_r),\ \tau_2,\ \tau_2^{\downarrow\square}),\ \sigma_f \Downarrow^{L_1} v_r',\ \sigma_f,\ \varnothing,\ 1 + \texttt{cost}_{\texttt{conv}}(\tau_2, \tau_2^{\downarrow\square}) \tag{D1}$$

and

$$(v_{ir},\ v_{cr},\ v_r',\ (\sigma_f,\ \sigma_c,\ \emptyset,\ m)) \in \mathcal{V}[\![\tau_2^{\downarrow\square}]\!] \tag{D2}$$

We can now show the goals

1. Combining eq. (A), eq. (C1) and eq. (D1) we can obtain

$$\underline{\mathbf{deep!}}(e,\ \tau),\ \sigma_i \Downarrow^{L_1} (v'_l, v'_r),\ \sigma_f,\ \varnothing,\ c + 3 + \mathtt{cost}_{\mathtt{conv}}(\tau_1, \tau_1^{\downarrow\Box}) + \mathtt{cost}_{\mathtt{conv}}(\tau_2, \tau_2^{\downarrow\Box})$$

2. We pick an arbitrary β and we instantiate eq. (C2) and eq. (D2) with it. We can easily derive

$$((v_{il}, v_{ir}),\ (v_{cl}, v_{cr}),\ (v'_l, v'_{cr}),\ (\sigma_f,\ \sigma_c,\ \beta,\ m)) \in \mathcal{V}[\![\tau_2^{\downarrow\Box} \times \tau_1^{\downarrow\Box}]\!]$$

that proves the goal.

□

Lemma 24 (Referencing)
Assume that the following hold

$$e_t,\ \sigma_i \Downarrow^{L_1} v_t,\ \sigma_f,\ \varnothing,\ c \tag{A}$$

$$(v_i,\ v_c,\ v_t,\ (\sigma_f,\ \sigma_c,\ \beta,\ m)) \in \mathcal{V}[\![\tau]\!] \tag{B}$$

Then exists $\sigma'_f \sqsupseteq^{L_1} \sigma_f$

1. $\underline{\mathbf{conv}}(e_t,\ \tau^{\downarrow\Box},\ \tau),\ \sigma_i \Downarrow^{L_1} v'_t,\ \sigma'_f,\ \varnothing,\ c + \mathtt{cost}_{\mathtt{conv}}(\tau^{\downarrow\Box}, \tau)$
2. $(v_i,\ v_c,\ v'_t,\ (\sigma'_f,\ \emptyset,\ \emptyset,\ m)) \in \mathcal{V}[\![\tau]\!]$

Proof. The proof is by induction on τ. It resembles the proof of lemma 23. □

Lemma 25 (Stable value lemma)
Assume that $\Delta; \Phi \vdash \tau \sqsubseteq \tau^{\downarrow\Box}$, $\varphi \in \mathcal{D}[\![\Delta]\!]$, $\models \varphi\Phi$ and

$$(v_i,\ v_c,\ v_t,\ (\sigma_i,\ \sigma_c,\ \beta,\ m)) \in \mathcal{V}[\![\varphi\tau]\!]$$

Then

$$(v_i,\ v_c,\ v_t,\ (\sigma_i,\ \emptyset,\ \emptyset,\ m)) \in \mathcal{V}[\![\varphi\tau]\!]$$

.

Proof. By induction on the size of τ and case analysis on the subtyping judgment. □

Lemma 26 (Stable context lemma)
Assume that $\Delta; \Phi \vdash \Gamma(x) \sqsubseteq \Gamma(x)^{\downarrow\Box}$, $\varphi \in \mathcal{D}[\![\Delta]\!]$, $\models \varphi\Phi$ and

$$(\theta_i,\ \theta_c,\ \theta_t,\ (\sigma_i,\ \sigma_c,\ \beta,\ m)) \in \mathcal{G}[\![\varphi\Gamma]\!]$$

Then $\beta = \emptyset$.

Proof. By induction on the size of Γ and using lemma 25. □

Lemma 27 (Subtyping translation evaluation)
If

$$\Delta; \Phi \vdash^{\kappa}_{\epsilon} \tau \sqsubseteq \tau' \hookrightarrow \underline{\mathbf{g}}$$

$$e,\ \sigma_1 \Downarrow^{L} v,\ \sigma_2,\ D_1,\ c_1$$

and

$$\underline{\mathbf{g}}(v),\ \sigma_2 \Downarrow^{L} v',\ \sigma_3,\ D_2,\ c_2$$

then

$$\underline{\mathbf{g}}(e),\ \sigma_1 \Downarrow^{L} v',\ \sigma_3,\ D_2,\ c_2$$

Proof. It follows easily by induction on the subtyping derivation. □

Lemma 28 (Subtyping soundness - Unary interpretation)
Assume that $\varphi \in \mathcal{D}[\![\Delta]\!]$ and $\models \varphi\Phi$. The following hold:

1. If $\Delta; \Phi \vdash_{\epsilon}^{\kappa} \tau \sqsubseteq \tau' \hookrightarrow \underline{\mathbf{g}}$ and $(v_s,\ v_t,\ (\sigma_i,\ m)) \in \mathcal{V}(\!|\varphi\tau|\!)_L$ then
$$(v_s,\ \underline{\mathbf{g}}(v_t),\ (\sigma_i,\ m)) \in \mathcal{E}(\!|\varphi\tau|\!)_L^{\kappa}$$

2. If $\Delta; \Phi_A \vdash_{\epsilon}^{\kappa'} A \sqsubseteq A' \hookrightarrow \underline{\mathbf{g}}$ and $(v_s,\ v_t,\ (\sigma_i,\ m)) \in \mathcal{V}_A(\!|\varphi A|\!)_L$ then
$$(v_s,\ \underline{\mathbf{g}}(v_t),\ (\sigma_i,\ m)) \in \mathcal{E}(\!|\varphi\tau|\!)_L^{\kappa}$$

3. If $\Delta; \Phi \vdash_{\epsilon}^{\kappa'} \tau \sqsubseteq \tau' \hookrightarrow \underline{\mathbf{g}}$ and $(e_s,\ e_t,\ (\sigma_i,\ m)) \in \mathcal{E}(\!|\varphi\tau|\!)_L^{\kappa'}$ then
$$(e_s,\ \underline{\mathbf{g}}(e_t),\ (\sigma_i,\ m)) \in \mathcal{E}(\!|\varphi\tau'|\!)_L^{\kappa\ \hat{+}\ \kappa'}$$

Proof. □

Lemma 29 (Subtyping soundness - Binary interpretation)
Assume that $\varphi \in \mathcal{D}[\![\Delta]\!]$ and $\models \varphi\Phi$. The following hold:

1. If $\Delta; \Phi_A \vdash_{\epsilon}^{\kappa} A \sqsubseteq A' \hookrightarrow \underline{\mathbf{g}}$ and $(v_i,\ v_c,\ v_t,\ (\sigma_i,\ \sigma_c,\ \beta,\ m)) \in \mathcal{V}_A[\![\varphi A]\!]$ then
$$(v_i,\ v_c,\ \underline{\mathbf{g}}(v_t),\ (\sigma_f,\ \sigma_f',\ \beta',\ m)) \in \mathcal{E}[\![\varphi A']\!]^{\kappa}$$

2. If $\Delta; \Phi \vdash_{\epsilon}^{\kappa'} \tau \sqsubseteq \tau' \hookrightarrow \underline{\mathbf{g}}$ and $(v_i,\ v_c,\ v_t,\ (\sigma_i,\ \sigma_c,\ \beta,\ m)) \in \mathcal{V}[\![\varphi\tau]\!]$ then
$$(v_i,\ v_c,\ \underline{\mathbf{g}}(v_t),\ (\sigma_f,\ \sigma_f',\ \beta',\ m)) \in \mathcal{E}[\![\varphi A']\!]^{\kappa}$$

3. If $\Delta; \Phi \vdash_{\epsilon}^{\kappa'} \tau \sqsubseteq \tau' \hookrightarrow \underline{\mathbf{g}}$ and $(e_i,\ e_c,\ e_t,\ (\sigma_i,\ \sigma_c,\ \beta,\ m)) \in \mathcal{E}[\![\varphi\tau]\!]^{\kappa'}$ then
$$(e_i,\ e_c,\ \underline{\mathbf{g}}(e_t),\ (\sigma_i,\ \sigma_c,\ \beta,\ m)) \in \mathcal{E}[\![\varphi\tau']\!]^{\kappa\ \hat{+}\ \kappa'}$$

Proof. We will prove the statements 1 and 2 by induction on the subtyping derivation. The cases where the translation is the identity are trivial, thus omitted.

- $$\frac{\Delta; \Phi \vdash_{\delta}^{\kappa_1} \tau_1' \sqsubseteq \tau_1 \hookrightarrow \underline{\mathbf{g_1}} \qquad \Delta; \Phi \vdash_{\delta}^{\kappa_2} \tau_2 \sqsubseteq \tau_2' \hookrightarrow \underline{\mathbf{g_2}} \qquad \Delta; \Phi \models \kappa \mathrel{\hat{+}} \kappa_1 \mathrel{\hat{+}} \kappa_2 \mathrel{\hat{+}} (\delta = \mathbb{C}) \mathrel{?} 1 : \ 0 \leq \kappa'}{\Delta; \Phi_A \vdash_{\epsilon}^{0} \tau_1 \xrightarrow{\delta(\kappa)} \tau_2 \sqsubseteq \tau_1' \xrightarrow{\delta(\kappa')} \tau_2' \hookrightarrow \underline{\lambda} e.\mathtt{fix}\ f(x).\, \underline{\mathbf{g_2}}(e\ \underline{\mathbf{g_1}}(x))} \to 2$$

There are two possible cases.

▶ $\delta = \mathbb{S}$

We pick arbitrary $j_i < m$, v_i', $\sigma_i' \sqsupseteq^{L_1} \sigma_i$, $\sigma_c' \sqsupseteq^{L_2} \sigma_c$, $\beta' \geq \beta$ such that
$$v_i \Downarrow v_i',\ j_i'$$
$$\mathtt{dom}(\beta') \setminus \mathtt{dom}(\beta) \subseteq \mathtt{dom}(\sigma_i') \setminus \mathtt{dom}(\sigma_i)$$
and
$$\mathtt{dom}'(\beta') \setminus \mathtt{dom}'(\beta) \subseteq \mathtt{dom}(\sigma_c') \setminus \mathtt{dom}(\sigma_c)$$

Since v_i is a value we can derive $v_i' = v_i$ and $j_i = 0$. We can now prove the goals.

1. We can easily show that

$$v_c \Downarrow v_c,\ 0$$

2. We can also show that

$$\lambda x.\, \underline{\mathbf{g_2}}(v_t\ \underline{\mathbf{g_1}}(x)),\ \sigma_i' \Downarrow^{L_1} \lambda x.\, \underline{\mathbf{g_2}}(v_t\ \underline{\mathbf{g_1}}(x)),\ \sigma_i',\ \varnothing,\ 0$$

3. We can also show that

$$\varnothing,\ \sigma_c',\ \sigma_i',\ \beta' \rightsquigarrow \sigma_c',\ \beta,\ 0$$

4. We trivially show that $0 \leq 0$
5. This goal is trivial since $\mathtt{dom}(\beta) \setminus \mathtt{dom}(\beta) = \emptyset$ and $\mathtt{dom}'(\beta) \setminus \mathtt{dom}'(\beta) = \emptyset$
6. It remains to show that

$$(v_i,\ v_c,\ \lambda x.\, \underline{\mathbf{g_2}}(v_t\ \underline{\mathbf{g_1}}(x)),\ (\sigma_i',\ \sigma_c',\ \beta',\ m)) \in \mathcal{V}[\![\varphi\tau_1' \xrightarrow{\mathbb{S}(\varphi\kappa')} \varphi\tau_2']\!]$$

From the hypothesis we know that

$$(v_i,\ v_c,\ v_t,\ (\sigma_i,\ \sigma_c,\ \beta,\ m)) \in \mathcal{V}[\![\varphi\tau_1 \xrightarrow{\mathbb{S}(\varphi\kappa)} \varphi\tau_2]\!] \tag{A}$$

We can derive that $v_i = \mathtt{fix}\ f(x).\, e_i'$, $v_c = \mathtt{fix}\ f(x).\, e_c'$ and $v_i = \mathtt{fix}\ f(x).\, e_t'$ for some e_i', e_c' and e_t'. We pick arbitrary $j < m$, $\sigma_i'' \sqsupseteq^{L_1} \sigma_i'$, $\sigma_c'' \sqsupseteq^{L_2} \sigma_c'$, $\beta'' \geq \beta'$, v_i', v_c' and v_t', such that

$$(v_i',\ v_c',\ v_t',\ (\sigma_i'',\ \sigma_c'',\ \beta'',\ j)) \in \mathcal{V}[\![\varphi\tau_1']\!]$$

$$\mathtt{dom}(\beta'') \setminus \mathtt{dom}(\beta') \subseteq \mathtt{dom}(\sigma_i'') \setminus \mathtt{dom}(\sigma_i')$$

and

$$\mathtt{dom}'(\beta'') \setminus \mathtt{dom}'(\beta') \subseteq \mathtt{dom}(\sigma_c'') \setminus \mathtt{dom}(\sigma_c')$$

We have to show that

$$([x \mapsto v_i', f \mapsto \ldots]e_i',\ [x \mapsto v_c', f \mapsto \ldots]e_c',\ [x \mapsto v_t']\underline{\mathbf{g_2}}((\mathtt{fix}\ f(x).\, e_t')\ \underline{\mathbf{g_1}}(x)),\ (\sigma_i'',\ \sigma_c'',\ \beta'',\ j)) \in \mathcal{E}[\![\varphi\tau_2']\!]^{\varphi\kappa'}$$

or equivalently

$$([x \mapsto v_i', f \mapsto \ldots]e_i',\ [x \mapsto v_c', f \mapsto \ldots]e_c',\ \underline{\mathbf{g_2}}((\mathtt{fix}\ f(x).\, e_t')\ \underline{\mathbf{g_1}}(v_t')),\ (\sigma_i'',\ \sigma_c'',\ \beta'',\ j)) \in \mathcal{E}[\![\varphi\tau_2']\!]^{\varphi\kappa'}$$

We pick arbitrary $j' < j$, $\sigma_i''' \sqsupseteq^{L_1} \sigma_i''$, $\sigma_c''' \sqsupseteq^{L_2} \sigma_c''$, $\beta''' \geq \beta''$, v_{if} and j_i such that

$$[x \mapsto v_i', f \mapsto \mathtt{fix}\ f(x).\, e_i']e_i' \Downarrow v_{if},\ j_i \tag{1}$$

$$\mathtt{dom}(\beta''') \setminus \mathtt{dom}(\beta'') \subseteq \mathtt{dom}(\sigma_i''') \setminus \mathtt{dom}(\sigma_i'')$$

and

$$\mathtt{dom}'(\beta''') \setminus \mathtt{dom}(\beta'') \subseteq \mathtt{dom}(\sigma_c''') \setminus \mathtt{dom}(\sigma_c'')$$

Using lemma lemma 15 (note that the conditions for the stores and the bijection hold) we can derive

$$(v_i',\ v_c',\ v_t',\ (\sigma_i''',\ \sigma_c''',\ \beta''',\ j)) \in \mathcal{V}[\![\tau_1']\!]$$

From the induction hypothesis applied on the first premise and the above statement we can obtain v_t'', $\sigma_1 \sqsupseteq^{L_1} \sigma_i'''$, $\sigma_1' \sqsupseteq^{L_1} \sigma_c'''$, c_1, $\beta_1 \geq \beta'''$ and c_1' such that

$$\underline{\mathbf{g_1}}(v_t'),\ \sigma_i''' \Downarrow^{L_1} v_t'',\ \sigma_1,\ D_1,\ c_1 \tag{B1}$$

$$D_1,\ \sigma_c''',\ \sigma_f,\ \beta''' \rightsquigarrow \sigma_1',\ \beta_1,\ c_1' \tag{B2}$$

$$c_1' \leq \kappa_1 \tag{B3}$$

$$\begin{gathered}\mathtt{dom}(\beta_1) \setminus \mathtt{dom}(\beta''') \subseteq \mathtt{dom}(\sigma_1) \setminus \mathtt{dom}(\sigma_i''') \\ \wedge \\ \mathtt{dom}'(\beta_1) \setminus \mathtt{dom}'(\beta''') \subseteq \mathtt{dom}(\sigma_1') \setminus \mathtt{dom}(\sigma_c''')\end{gathered} \tag{B4}$$

and

$$(v_i',\ v_c',\ v_t'',\ (\sigma_1,\ \sigma_1',\ \beta_1,\ j)) \in \mathcal{V}[\![\varphi\tau_1]\!] \tag{B5}$$

We can instantiate eq. (A) with eq. (B5) (note that the conditions for the stores and the bijection hold) in order to derive that

$$([x \mapsto v_i', f \mapsto \ldots]e_i',\ [x \mapsto v_c', f \mapsto \ldots]e_c',\ [x \mapsto v_t'', f \mapsto \ldots]e_t',\ (\sigma_1',\ \sigma_1',\ \beta_1,\ j)) \in \mathcal{E}[\![\varphi\tau_2]\!]^{\varphi\kappa}$$

Form the induction hypothesis for the second premise and the above statement can derive that

$$([x \mapsto v_i', f \mapsto \ldots]e_i',\ [x \mapsto v_c', f \mapsto \ldots]e_c',\ \underline{\mathbf{g_2}}([x \mapsto v_t'', f \mapsto \ldots]e_t'),\ (\sigma_1,\ \sigma_1',\ \beta_1,\ j)) \in \mathcal{E}[\![\varphi\tau_2']\!]^{\varphi\kappa+\varphi\kappa_2} \tag{C}$$

We instantiate eq. (C) with the above, eq. (1), σ_1, σ_1' and β_1 (note that the preconditions for the stores and the bijection hold) and we obtain v_{cf}, j_c, v_{tf}, σ_2, D_2, c_2, σ_2', β_2, c_2' such that:

$$[x \mapsto v_c, f \mapsto \mathtt{fix}\ f(x).\, e_c']e_c' \Downarrow v_{cf},\ j_c \tag{C1}$$

$$\underline{\mathbf{g_2}}([x \mapsto v_t'', f \mapsto \mathtt{fix}\ f(x).\, e_t']e_t'),\ \sigma_f \Downarrow^{L_1} v_{tf},\ \sigma_1,\ D,\ c_2 \tag{C2}$$

$$D_2,\ \sigma_1',\ \sigma_2,\ \beta_1 \rightsquigarrow \sigma_2',\ \beta_2,\ c_2' \tag{C3}$$

$$c_2' \leq \varphi\kappa_2 + \varphi\kappa \tag{C4}$$

$$\begin{gathered}\texttt{dom}(\beta_2) \setminus \texttt{dom}(\beta_1) \subseteq \texttt{dom}(\sigma_2) \setminus \texttt{dom}(\sigma_1) \\ \wedge \\ \texttt{dom}'(\beta_2) \setminus \texttt{dom}'(\beta_1) \subseteq \texttt{dom}(\sigma_2') \setminus \texttt{dom}(\sigma_1')\end{gathered} \tag{C5}$$

and

$$(v_{if},\ v_{cf},\ v_{tf},\ (\sigma_2,\ \sigma_2',\ \beta_2,\ m - j_i)) \in \mathcal{V}[\![\varphi\tau_2']\!] \tag{C6}$$

Using eq. (B1), eq. (C2) and lemma 27 we can show that

$$\underline{\mathbf{g_2}}((\texttt{fix}\ f(x).\, e_t')\ \underline{\mathbf{g_1}}(v_t)),\ \sigma_i'' \Downarrow^{L_1} v_{tf},\ \sigma_2,\ D_1 + D_2,\ c_1 + c_2 + 1$$

We can apply lemma 13 to eq. (B2) and eq. (C3) and derive

$$D_1 + D_2,\ \sigma_c'',\ \sigma_2,\ \beta'' \rightsquigarrow \sigma_2',\ \beta_2,\ c_1' + c_2'$$

From eq. (B3) and eq. (C4)

$$c_1' + c_2' \leq \varphi\kappa + \varphi\kappa_1 + \varphi\kappa_2$$

The result then follows from the above and eq. (C1), eq. (C5) and eq. (C6).

▶ $\delta = \mathbb{C}$

We pick arbitrary $j_i < m$, v_i', $\sigma_i' \sqsupseteq^{L_1} \sigma_i$, $\sigma_c' \sqsupseteq^{L_2} \sigma_c$, $\beta' \geq \beta$ such that

$$v_i \Downarrow v_i',\ j_i'$$

$$\texttt{dom}(\beta') \setminus \texttt{dom}(\beta) \subseteq \texttt{dom}(\sigma_i') \setminus \texttt{dom}(\sigma_i)$$

and

$$\texttt{dom}'(\beta') \setminus \texttt{dom}'(\beta) \subseteq \texttt{dom}(\sigma_c') \setminus \texttt{dom}(\sigma_c)$$

Since v_i is a value we can derive $v_i' = v_i$ and $j_i = 0$. We can now prove the goals.

1. We can easily show that
$$v_c \Downarrow v_c,\ 0$$
2. We can also show that
$$\lambda x.\, \texttt{let}\ v\ =\ v_t\ \underline{\mathbf{g_1}}(x)\ \texttt{in}\ \underline{\mathbf{g_2}}(v),\ \sigma_i' \Downarrow^{L_1} \lambda x.\, \texttt{let}\ v\ =\ v_t\ \underline{\mathbf{g_1}}(x)\ \texttt{in}\ \underline{\mathbf{g_2}}(v),\ \sigma_i',\ \varnothing,\ 0$$
3. We can also show that
$$\varnothing,\ \sigma_c',\ \sigma_i',\ \beta' \rightsquigarrow \sigma_c',\ \beta',\ 0$$
4. We trivially show that $\models 0 \dot{\leq} 0$
5. This goal is trivial since $\texttt{dom}(\beta') \setminus \texttt{dom}(\beta') = \emptyset$ and $\texttt{dom}'(\beta') \setminus \texttt{dom}'(\beta') = \emptyset$

6. It remains to show that

$$(v_i,\ v_c,\ \lambda x.\,\underline{\mathbf{g_2}}(v_t\ \underline{\mathbf{g_1}}(x)),\ (\sigma_i',\ \sigma_c',\ \beta,\ m)) \in \mathcal{V}\llbracket \varphi\tau_1' \xrightarrow{\mathbb{C}(\varphi\kappa')} \varphi\tau_2' \rrbracket$$

From the hypothesis we know that

$$(v_i,\ v_c,\ v_t,\ (\sigma_i,\ \sigma_c,\ \beta,\ m)) \in \mathcal{V}\llbracket \varphi\tau_1 \xrightarrow{\mathbb{C}(\varphi\kappa)} \varphi\tau_2 \rrbracket$$

We can unfold the definition and obtain

$$(v_i,\ v_t,\ (\sigma_i,\ j)) \in \mathcal{V}\llparenthesis \varphi\tau_1 \xrightarrow{\mathbb{C}(\varphi\kappa)} \varphi\tau_2 \rrparenthesis_{L_1} \tag{A}$$

$$\forall m,\ (v_c,\ \beta(v_t),\ (\sigma_c + \sigma_i,\ j)) \in \mathcal{V}\llparenthesis \varphi\tau_1 \xrightarrow{\mathbb{C}(\varphi\kappa)} \varphi\tau_2 \rrparenthesis_{L_2} \tag{B}$$

Also using the premises we can construct a derivation for

$$\Delta; \Phi_A \vdash^0_{\mathbb{C}} \tau_1 \xrightarrow{\delta(\kappa)} \tau_2 \sqsubseteq \tau_1' \xrightarrow{\delta(\kappa')} \tau_2' \hookrightarrow \underline{\lambda} e.\texttt{fix}\ f(x).\,\underline{\mathbf{g_2}}(e\ \underline{\mathbf{g_1}}(x)) \tag{C}$$

We apply lemma lemma 28 to eq. (A) and eq. (C) and derive

$$(v_i,\ \lambda x.\,\underline{\mathbf{g_2}}(v_t\ \underline{\mathbf{g_1}}(x)),\ (\sigma_i,\ j)) \in \mathcal{E}\llparenthesis \varphi\tau_1 \xrightarrow{\mathbb{C}(\varphi\kappa')} \varphi\tau_2 \rrparenthesis^0_{L_1}$$

Using the above we can easily show that

$$(v_i,\ \lambda x.\,\underline{\mathbf{g_2}}(v_t\ \underline{\mathbf{g_1}}(x)),\ (\sigma_i,\ j)) \in \mathcal{V}\llparenthesis \varphi\tau_1 \xrightarrow{\mathbb{C}(\varphi\kappa')} \varphi\tau_2 \rrparenthesis_{L_1} \tag{A1}$$

Similarly, we apply lemma lemma 28 to eq. (B) and eq. (C) and we derive

$$\forall m,\ (v_i,\ \lambda x.\,\underline{\mathbf{g_2}}(\beta(v_t)\ \underline{\mathbf{g_1}}(x)),\ (\sigma_c + \sigma_i,\ j)) \in \mathcal{E}\llparenthesis \varphi\tau_1' \xrightarrow{\mathbb{C}(\varphi\kappa')} \varphi\tau_2' \rrparenthesis^0_{L_2}$$

Using the above we can easily show that

$$\forall m,\ (v_i,\ \lambda x.\,\texttt{let}\ v\ =\ \beta(v_t)\ \underline{\mathbf{g_1}}(x)\ \texttt{in}\ \underline{\mathbf{g_2}}(v),\ (\sigma_c + \sigma_i,\ j)) \in \mathcal{V}\llparenthesis \varphi\tau_1' \xrightarrow{\mathbb{C}(\varphi\kappa')} \varphi\tau_2' \rrparenthesis_{L_2} \tag{A2}$$

Using eq. (A1) and eq. (A2) we can show that

$$(v_i,\ v_c,\ v_t,\ (\sigma_i,\ \sigma_c,\ \beta,\ m)) \in \mathcal{V}\llbracket \varphi\tau_1' \xrightarrow{\mathbb{C}(\varphi\kappa')} \varphi\tau_2' \rrbracket$$

which proves the goal.

- $$\frac{\Delta t :: S; \Phi \vdash^{\kappa_1}_{\epsilon} \tau \sqsubseteq \tau' \hookrightarrow \underline{\mathbf{g}} \qquad \Delta; \Phi \models \kappa \mathrel{\hat{+}} \kappa_1 \mathrel{\hat{+}} ((\delta = \mathbb{C})\ \textbf{?}\ 1\ :\ 0) \leq \kappa'}{\Delta; \Phi \vdash^0_{\epsilon} \forall t \overset{\delta(\kappa)}{::} S.\ \tau \sqsubseteq \forall t \overset{\delta(\kappa')}{::} S.\ \tau' \hookrightarrow \underline{\lambda} e.\lambda x.\,\underline{\mathbf{g}}(e\ ())}\forall 2$$

The proof of this case is similar to the one of → 2.

- $$\frac{\Delta;\Phi \vdash^{\kappa_1}_{\epsilon} \tau_1 \sqsubseteq \tau_1' \hookrightarrow \underline{\mathbf{g_1}} \qquad \Delta;\Phi \vdash^{\kappa_2}_{\epsilon} \tau_2 \sqsubseteq \tau_2' \hookrightarrow \underline{\mathbf{g_2}} \qquad \Delta;\Phi \models \kappa \doteq \kappa_1 \hat{+} \kappa_2}{\Delta;\Phi \vdash^{\kappa}_{\epsilon} \tau_1 \times \tau_2 \sqsubseteq \tau_1' \times \tau_2' \hookrightarrow \underline{\lambda} e.(\underline{\mathbf{g_1}}(\texttt{fst } e), \underline{\mathbf{g_2}}(\texttt{snd } e))} \times$$

The proof of this case follows easily from the induction hypotheses for the two premises.

- $$\frac{\Delta;\Phi \models \kappa \doteq (\epsilon = \mathbb{C}) \mathrel{\mathbf{?}} 1 \hat{+} \hat{\texttt{max}}(\texttt{cost}_{\texttt{conv}}(\tau_1, \tau_1^{\downarrow\Box}), \texttt{cost}_{\texttt{conv}}(\tau_2, \tau_2^{\downarrow\Box})) \;:\; 0}{\Delta;\Phi \vdash^{\kappa}_{\epsilon} (\tau_1 + \tau_2)^{\Box} \sqsubseteq (\tau_1^{\downarrow\Box} + \tau_2^{\downarrow\Box})^{\Box} \hookrightarrow \underline{\lambda} e.\texttt{case}(e, x.\texttt{inl } \underline{\mathbf{conv}}(x,\ \tau_1,\ \tau_1^{\downarrow\Box}), y.\texttt{inr } \underline{\mathbf{conv}}(y,\ \tau_2,\ \tau_2^{\downarrow\Box}))} \mathbf{+1}$$

From the hypothesis we know that

$$(v_i,\ v_c,\ v_t,\ (\sigma_i,\ \sigma_c,\ \beta,\ m)) \in \mathcal{V}[\![(\varphi\tau_1 + \varphi\tau_2)^{\Box}]\!]$$

From this we derive

$$(v_i,\ v_c,\ v_t,\ (\sigma_i,\ \emptyset,\ \emptyset,\ j)) \in \mathcal{V}_A[\![\varphi\tau_1 + \varphi\tau_2]\!]$$

There are two possible cases, either $v_i = \texttt{inl } v_i'$, $v_c = \texttt{inl } v_c'$ and $v_t = \texttt{inl } v_t'$ or $v_i = \texttt{inr } v_i'$, $v_c = \texttt{inr } v_c'$ and $v_t = \texttt{inr } v_t'$. The two cases are symmetric, so we will consider only the first one. From the definition of the logical relation we obtain

$$(v_i',\ v_c',\ v_t',\ (\sigma_i,\ \emptyset,\ \emptyset,\ m)) \in \mathcal{V}[\![\varphi\tau_1]\!]$$

We pick arbitrary $j_i < m$, v_i'', $\sigma_i' \sqsupseteq^{L_1} \sigma_i$, $\sigma_c' \sqsupseteq^{L_2} \sigma_c$, $\beta' \geq \beta$ such that

$$v_i' \Downarrow v_i'',\ j_i$$

$$\texttt{dom}(\beta') \setminus \texttt{dom}(\beta) \subseteq \texttt{dom}(\sigma_i') \setminus \texttt{dom}(\sigma_i)$$

and

$$\texttt{dom}'(\beta') \setminus \texttt{dom}'(\beta) \subseteq \texttt{dom}(\sigma_c') \setminus \texttt{dom}(\sigma_c)$$

Since v_i' is a value we obtain $v_i' = v_i''$ and $j_i = 0$. Since v_t is a value we obtain

$$v_t,\ \sigma_i' \Downarrow^{L_1} v_t,\ \sigma_i',\ \varnothing,\ 0$$

We can apply lemma 23 to the two statements above and obtain

$$\underline{\mathbf{conv}}(v_t',\ \tau_1,\ \tau_1^{\downarrow\Box}),\ \sigma_i' \Downarrow^{L_1} v_t'',\ \sigma_i',\ \varnothing,\ \texttt{cost}_{\texttt{conv}}(\tau_1, \tau_1^{\downarrow\Box}) \tag{A}$$

and

$$(v_i',\ v_c',\ v_t'',\ (\sigma_i,\ \emptyset,\ \emptyset,\ m)) \in \mathcal{V}[\![\varphi\tau_1]\!] \tag{B}$$

We can now show the goals

1. Since v_c' is a value we can easily derive $\texttt{inl } v_c' \Downarrow \texttt{inl } v_c',\ 0$

2. Using eq. (A) we can show

$$\texttt{case}(\texttt{inl}\; v'_t, x.\texttt{inl}\; \underline{\mathbf{conv}}(x,\ \tau_1,\ \tau_1^{\downarrow\Box}), x.\texttt{inr}\; \underline{\mathbf{conv}}(x,\ \tau_2,\ \tau_2^{\downarrow\Box})),\ \sigma'_i \Downarrow^{L_1} \texttt{inl}\; v''_t,\ \sigma'_i,\ \varnothing,\ 1{+}\texttt{cost}_{\texttt{conv}}(\tau_1, \tau_1^{\downarrow\Box})$$

3. We can trivially show that

$$\varnothing,\ \sigma'_c,\ \sigma'_i,\ \beta' \rightsquigarrow \sigma'_c,\ \beta',\ 0$$

4. We can trivially show that $\models 0 \mathrel{\dot\leq} 0$
5. This goal is trivial since $\texttt{dom}(\beta') \setminus \texttt{dom}(\beta') = \emptyset$ and $\texttt{dom}'(\beta') \setminus \texttt{dom}'(\beta') = \emptyset$
6. We have to show that

$$(\texttt{inl}\; v'_i,\ \texttt{inl}\; v'_c,\ \sigma'_i,\ (\sigma'_c,\ \beta',\ m,\ \ \in))\mathcal{V}[\![(\varphi\tau_1^{\downarrow\Box} + \varphi\tau_2^{\downarrow\Box})^{\Box}]\!]$$

which follows from the definition of the logical relation and eq. (B).

- $$\frac{\Delta; \Phi \models \kappa \doteq (\epsilon = \mathbb{C})\ \mathbf{?}\ 1 \mathbin{\hat{+}} \hat{\texttt{max}}(\texttt{cost}_{\texttt{conv}}(\tau_1^{\downarrow\Box}, \tau_1), \texttt{cost}_{\texttt{conv}}(\tau_2^{\downarrow\Box}, \tau_2))\ :\ 0}{\Delta; \Phi \vdash_{\epsilon}^{\kappa} (\tau_1^{\downarrow\Box} + \tau_2^{\downarrow\Box})^{\Box} \sqsubseteq (\tau_1 + \tau_2)^{\Box} \hookrightarrow \underline{\lambda} e.\texttt{case}(e, x.\underline{\mathbf{conv}}(x,\ \tau_1^{\downarrow\Box},\ \tau_1), y.\underline{\mathbf{conv}}(y,\ \tau_2^{\downarrow\Box},\ \tau_2))}\ \mathbf{+2}$$

From the hypothesis we know that

$$(v_i,\ v_c,\ v_t,\ (\sigma_i,\ \sigma_c,\ \beta,\ j)) \in \mathcal{V}[\![(\varphi\tau_1^{\downarrow\Box} + \varphi\tau_2^{\downarrow\Box})^{\Box}]\!]$$

From this we derive

$$(v_i,\ v_c,\ v_t,\ (\sigma_i,\ \emptyset,\ \emptyset,\ j)) \in \mathcal{V}_A[\![\varphi\tau_1^{\downarrow\Box} + \varphi\tau_2^{\downarrow\Box}]\!]$$

There are two possible cases, either $v_i = \texttt{inl}\; v'_i$, $v_c = \texttt{inl}\; v'_c$ and $v_t = \texttt{inl}\; v'_t$ or $v_i = \texttt{inr}\; v'_i$, $v_c = \texttt{inr}\; v'_c$ and $v_t = \texttt{inr}\; v'_t$, for some v'_i, v'_c and v'_t. The two cases are symmetric, so we will consider only the first one. From the definition of the logical relation we obtain

$$(v'_i,\ v'_c,\ v'_t,\ (\sigma_i,\ \emptyset,\ \emptyset,\ j)) \in \mathcal{V}[\![\varphi\tau_1^{\downarrow\Box}]\!]$$

We pick arbitrary $j_i < m$, v''_i, $\sigma'_i \sqsupseteq^{L_1} \sigma_i$, $\sigma'_c \sqsupseteq^{L_2} \sigma_c$, $\beta' \geq \beta$ such that

$$v'_i \Downarrow v''_i,\ j_i$$

$$\texttt{dom}(\beta') \setminus \texttt{dom}(\beta) \subseteq \texttt{dom}(\sigma'_i) \setminus \texttt{dom}(\sigma_i)$$

and

$$\texttt{dom}'(\beta') \setminus \texttt{dom}'(\beta) \subseteq \texttt{dom}(\sigma'_c) \setminus \texttt{dom}(\sigma_c)$$

Since v'_i is a value we obtain $v'_i = v''_i$ and $j_i = 0$. Since v_t is a value we obtain

$$v_t,\ \sigma'_i \Downarrow^{L_1} v_t,\ \sigma'_i,\ \varnothing,\ 0$$

We can apply lemma 24 to the two statements above and obtain $\sigma_f \sqsupseteq^{L_1} \sigma_i'$ such that

$$\underline{\mathbf{conv}}(v_t',\ \tau_1^{\downarrow\square},\ \tau_1),\ \sigma_i' \Downarrow^{L_1} v_t'',\ \sigma_f,\ \varnothing,\ \mathtt{cost}_{\mathtt{conv}}(\tau_1^{\downarrow\square}, \tau_1) \tag{A}$$

and

$$(v_i',\ v_c',\ v_t'',\ (\sigma_f,\ \emptyset,\ \emptyset,\ m)) \in \mathcal{V}[\![\varphi\tau_1]\!] \tag{B}$$

We can now show the goals

1. Since v_c' is a value we can easily derive $\mathtt{inl}\ v_c' \Downarrow \mathtt{inl}\ v_c',\ 0$
2. Using eq. (A) we can show

$$\begin{gathered}\mathtt{case}(\mathtt{inl}\ v_t', x.\mathtt{inl}\ \underline{\mathbf{conv}}(x,\ \tau_1^{\downarrow\square},\ \tau_1), x.\mathtt{inr}\ \underline{\mathbf{conv}}(x,\ \tau_2^{\downarrow\square},\ \tau_2)),\ \sigma_i' \\ \Downarrow^{L_1} \\ \mathtt{inl}\ v_t'',\ \sigma_f,\ \varnothing,\ 1 + \mathtt{cost}_{\mathtt{conv}}(\tau_1^{\downarrow\square}, \tau_1)\end{gathered}$$

3. We can trivially show that

$$\varnothing,\ \sigma_c',\ \sigma_i',\ \beta' \rightsquigarrow \sigma_c',\ \beta',\ 0$$

4. We can trivially show that $\models 0 \mathrel{\dot{\leq}} 0$
5. This goal is trivial since $\mathtt{dom}(\beta') \setminus \mathtt{dom}(\beta') = \emptyset$ and $\mathtt{dom}'(\beta') \setminus \mathtt{dom}'(\beta') = \emptyset$
6. We have to show that

$$(\mathtt{inl}\ v_i',\ \mathtt{inl}\ v_c',\ \sigma_f,\ (\sigma_c',\ \beta',\ m,\ \ \in))\mathcal{V}[\![(\varphi\tau_1 + \varphi\tau_2)^{\square}]\!]$$

   or, equivalently,

$$(\mathtt{inl}\ v_i',\ \mathtt{inl}\ v_c',\ \sigma_f,\ (\emptyset,\ \emptyset,\ m,\ \ \in))\mathcal{V}[\![(\varphi\tau_1 + \varphi\tau_2)^{\square}]\!]$$

   which follows from the definition of the logical relation and eq. (B).

- $$\frac{\Delta;\Phi \models n \doteq n' \qquad \Delta;\Phi \models \alpha \doteq \alpha' \qquad \Delta;\Phi \vdash^{\kappa'}_{\mathbb{S}} \tau \sqsubseteq \tau' \hookrightarrow \underline{\mathbf{g}} \qquad \Delta;\Phi \models \kappa \doteq \alpha \mathbin{\hat{\cdot}} \kappa'}{\Delta;\Phi \vdash^{\kappa}_{\mathbb{S}} \mathtt{list}\,[n]^{\alpha}\ \tau \sqsubseteq \mathtt{list}\,[n']^{\alpha'}\ \tau' \hookrightarrow \underline{\lambda} e.\mathtt{map}\ \lambda z.\,\mathtt{case}(z, x.\mathtt{inl}\ \underline{\mathbf{conv}}(\mathtt{drop}(\underline{\mathbf{g}}(\underline{\mathbf{conv}}(x,\ \tau^{\downarrow\square},\ \tau))),\ \tau',\ \tau'^{\downarrow\square}), y.\mathtt{inr}}$$

We will prove that for all $(v_i,\ v_c,\ v_t,\ (\sigma_i,\ \sigma_c,\ \beta,\ m)) \in \mathcal{V}[\![\tau^{\downarrow\square}]\!]$,

$$\begin{gathered}(\mathtt{inl}\ v_i,\ \mathtt{inl}\ v_c,\ (\lambda z.\,\mathtt{case}(z, x.\mathtt{inl}\ \underline{\mathbf{conv}}(\underline{\mathbf{g}}(\underline{\mathbf{conv}}(x,\ \tau^{\downarrow\square},\ \tau)),\ \tau,\ \tau^{\downarrow\square}), y.\mathtt{inr}\ \underline{\mathbf{g}}\ y))\ \mathtt{inl}\ v_t,\ (\sigma_i,\ \sigma_c,\ \beta,\ m)) \\ \in \mathcal{E}[\![\tau^{\downarrow\square} + \tau]\!]^{0}\end{gathered}$$

and for all $(v_i,\ v_c,\ v_t,\ (\sigma_i,\ \sigma_c,\ \beta,\ m)) \in \mathcal{V}[\![\tau]\!]$,

$$\begin{gathered}(\mathtt{inr}\ v_i,\ \mathtt{inr}\ v_c,\ (\lambda z.\,\mathtt{case}(z, x.\mathtt{inl}\ \underline{\mathbf{conv}}(\underline{\mathbf{g}}(\underline{\mathbf{conv}}(x,\ \tau^{\downarrow\square},\ \tau)),\ \tau,\ \tau^{\downarrow\square}), y.\mathtt{inr}\ \underline{\mathbf{g}}\ y))\ \mathtt{inr}\ v_t,\ (\sigma_i,\ \sigma_c,\ \beta,\ m)) \\ \in \mathcal{E}[\![\tau^{\downarrow\square} + \tau]\!]^{\kappa'}\end{gathered}$$

The result then follows from lemma 20.

To prove the first goal we pick arbitrary v_i, v_c, v_t, σ_i, σ_c, β and m such that

$$(v_i,\ v_c,\ v_t,\ (\sigma_i,\ \sigma_c,\ \beta,\ m)) \in \mathcal{V}[\![\tau^{\downarrow\Box}]\!]$$

We pick arbitrary $j_i < m$, v_i', $\sigma_i' \sqsupseteq^{L_1} \sigma_i$, $\sigma_c' \sqsupseteq^{L_2} \sigma_c$, $\beta' \geq \beta$ such that

$$\texttt{inl}\ v_i \Downarrow v_i',\ j_i$$

$$\texttt{dom}(\beta') \setminus \texttt{dom}(\beta) \subseteq \texttt{dom}(\sigma_i') \setminus \texttt{dom}(\sigma_i)$$

and

$$\texttt{dom}'(\beta') \setminus \texttt{dom}'(\beta) \subseteq \texttt{dom}(\sigma_c') \setminus \texttt{dom}(\sigma_c)$$

Since $\texttt{inl}\ v_i$ is a value we obtain $\texttt{inl}\ v_i = v_i'$ and $j_i = 0$. Since v_t is a value we obtain

$$v_t,\ \sigma_i' \Downarrow^{L_1} v_t,\ \sigma_i',\ \varnothing,\ 0$$

Using lemma 15 we can obtain

$$(v_i,\ v_c,\ v_t,\ (\sigma_i',\ \sigma_c',\ \beta',\ m)) \in \mathcal{V}[\![\tau^{\downarrow\Box}]\!]$$

We can apply lemma 23 to the two statements above and we obtain

$$\underline{\mathbf{conv}}(v_t,\ \tau^{\downarrow\Box},\ \tau),\ \sigma_i' \Downarrow^{L_1} v_t',\ \sigma_f,\ \varnothing,\ \texttt{cost}_{\texttt{conv}}(\tau^{\downarrow\Box}, \tau) \tag{A}$$

$$(v_i,\ v_c,\ v_t',\ (\sigma_f,\ \emptyset,\ \emptyset,\ m)) \in \mathcal{V}[\![\tau]\!] \tag{B}$$

We can apply the induction hypothesis and derive

$$(v_i,\ v_c,\ \underline{\mathbf{g}}(v_t'),\ (\sigma_f,\ \emptyset,\ \emptyset,\ m)) \in \mathcal{E}[\![\tau']\!]^{\kappa'}$$

We instantiate the above with 0 (note that $0 < m$), σ_f (note that $\sigma_f \sqsupseteq^{L_1} \sigma_i'$), $\emptyset$, $\emptyset$ and $v_i \Downarrow v_i,\ 0$, and we obtain v_c', j_c, v_t'', σ_1, D_1, c_1, σ_1', β_1, c_1' such that:

$$v_c \Downarrow v_c',\ j_c \tag{C1}$$

and since v_c is a value we immediately derive $v_c' = v_c$ and $j_c = 0$.

$$\underline{\mathbf{g}}(v_t'),\ \sigma_f \Downarrow^{L_1} v_t'',\ \sigma_1,\ D_1,\ c_1 \tag{C2}$$

$$D_1,\ \emptyset,\ \sigma_1,\ \emptyset \rightsquigarrow \sigma_1',\ \beta_1,\ c_1' \tag{C3}$$

and from the definition of change propagation we can show $\sigma'_1 = \emptyset$, $\beta_1 = \emptyset$ and $c'_1 = 0$.

$$\models 0 \mathrel{\dot{\leq}} \varphi\kappa' \tag{C4}$$

and

$$(v_i,\ v_c,\ v''_t,\ (\sigma_1,\ \emptyset,\ \emptyset,\ m)) \in \mathcal{V}[\![\tau']\!] \tag{C5}$$

Using eq. (C2), eq. (A) and lemma 27 we can show

$$\mathtt{drop}(\underline{\mathbf{g}}(\underline{\mathbf{conv}}(v_t,\ \tau^{\downarrow\square},\ \tau))),\ \sigma_f \Downarrow^{L_1} v''_t,\ \sigma_1,\ \varnothing,\ c_1 + \mathtt{cost}_{\mathtt{conv}}(\tau^{\downarrow\square}, \tau) + 1 \tag{D}$$

We can apply lemma 23 to eq. (D) and eq. (C5) and derive

$$\begin{array}{c} \underline{\mathbf{conv}}(\mathtt{drop}(\underline{\mathbf{g}}(\underline{\mathbf{conv}}(v_t,\ \tau^{\downarrow\square},\ \tau))),\ \tau',\ \tau'^{\downarrow\square}),\ \sigma_f \\ \Downarrow^{L_1} \\ v''_t,\ \sigma_1,\ \varnothing,\ c_1 + \mathtt{cost}_{\mathtt{conv}}(\tau^{\downarrow\square}, \tau) + \mathtt{cost}_{\mathtt{conv}}(\tau', \tau'^{\downarrow\square}) + 1 \end{array} \tag{E}$$

$$(v_i,\ v_c,\ v''_t,\ (\sigma_1,\ \sigma'_c,\ \beta',\ m)) \in \mathcal{V}[\![\tau'^{\downarrow\square}]\!] \tag{F}$$

We can now show the goals

1. We can easily derive that $\mathtt{inl}\ v_c \Downarrow \mathtt{inl}\ v_c,\ 0$
2. Using eq. (E)

$$\begin{array}{c} (\lambda z.\, \mathtt{case}(z, x.\mathtt{inl}\ \underline{\mathbf{conv}}(\mathtt{drop}(\underline{\mathbf{g}}(\underline{\mathbf{conv}}(x,\ \tau^{\downarrow\square},\ \tau))),\ \tau,\ \tau^{\downarrow\square}), y.\mathtt{inr}\ \underline{\mathbf{g}}\ y))\ \mathtt{inl}\ v_t,\ \sigma'_i \\ \Downarrow^{L_1} \\ \mathtt{inl}\ v''_t,\ \sigma_1,\ \varnothing,\ c_1 + \mathtt{cost}_{\mathtt{conv}}(\tau^{\downarrow\square}, \tau) + \mathtt{cost}_{\mathtt{conv}}(\tau, \tau^{\downarrow\square}) + 3 \end{array}$$

3. We can trivially show that

$$\varnothing,\ \sigma'_c,\ \sigma'_i,\ \beta' \rightsquigarrow \sigma'_c,\ \beta',\ 0$$

4. We can trivially show that $0 \leq 0$
5. This goal is trivial since $\mathtt{dom}(\beta') \setminus \mathtt{dom}(\beta') = \emptyset$ and $\mathtt{dom}'(\beta') \setminus \mathtt{dom}'(\beta') = \emptyset$
6. Using eq. (F) we can show that

$$(\mathtt{inl}\ v_i,\ \mathtt{inl}\ v_c,\ \mathtt{inl}\ v''_t,\ (\sigma_1,\ \sigma'_c,\ \beta',\ m)) \in \mathcal{V}[\![\tau'^{\downarrow\square} + \tau']\!]$$

We will now prove the second goal. We pick arbitrary v_i, v_c, v_t, σ_i, σ_c, β and m such that

$$(v_i,\ v_c,\ v_t,\ (\sigma_i,\ \sigma_c,\ \beta,\ m)) \in \mathcal{V}[\![\tau]\!]$$

We can apply the induction hypothesis and derive

$$(v_i,\ v_c,\ \underline{\mathbf{g}}(v_t),\ (\sigma_i,\ \sigma_i,\ \sigma_c,\ m)) \in \mathcal{E}[\![\tau']\!]^{\kappa'}$$

Using the above we can easily prove that

$$(\texttt{inr}\ v_i,\ \texttt{inr}\ v_c,\ (\lambda z.\,\texttt{case}(z, x., \ldots, , y.\texttt{inr}\ \underline{\mathbf{g}}\ y))\ \texttt{inr}\ v_t,\ (\sigma_i,\ \sigma_c,\ \beta,\ m)) \in \mathcal{E}[\![\tau'^{\downarrow\square} + \tau']\!]^{\kappa'}$$

That proves the goal.

- $$\dfrac{\Delta, t :: S; \Phi \vdash^{\kappa}_{\epsilon} \tau \sqsubseteq \tau' \hookrightarrow \underline{\mathbf{g}} \qquad t \notin FV(\Phi)}{\Delta; \Phi \vdash^{\kappa}_{\epsilon} \exists t.\ \tau \sqsubseteq \exists t.\ \tau' \hookrightarrow \underline{\mathbf{g}}}\ \exists$$

This case follows easily from the induction hypothesis.

- $$\dfrac{\Delta; \Phi \models \mu \leq \mu' \qquad \Delta; \Phi \models \kappa \doteq \texttt{cost}_{\texttt{conv}}((A)^{\mu}, (A)^{\mu'})}{\Delta; \Phi \vdash^{\kappa}_{\epsilon} (A)^{\mu} \sqsubseteq (A)^{\mu'} \hookrightarrow \underline{\lambda} e.\underline{\mathbf{conv}}(e,\ (A)^{\mu},\ (A)^{\mu'})}\ \mu$$

If $\mu = \mu'$ or both of μ, μ' are either $\mathbb{S}$ or $\square$ the translation function is the identity and the result is trivial. When $\mu = \square$ and $\mu' = \mathbb{C}$ the result follows using lemma lemma 24. The only remaining case is when $\mu = \mathbb{S}$ and $\mu' = \mathbb{C}$.

From the hypothesis we know

$$(v_i,\ v_c,\ v_t,\ (\sigma_i,\ \sigma_c,\ \beta,\ j)) \in \mathcal{V}[\![(\varphi A)^{\mathbb{S}}]\!]$$

thus

$$(v_i,\ v_c,\ v_t,\ (\sigma_i,\ \sigma_c,\ \beta,\ j)) \in \mathcal{V}_A[\![\varphi A]\!] \tag{A}$$

The goals are proved as follows

1. We can show that

   $$\underline{\mathbf{conv}}(v_t,\ (\varphi A)^{\mathbb{S}},\ (\varphi A)^{\mathbb{C}}),\ \sigma_i \Downarrow^{L_1} l,\ \sigma_i[l \mapsto v_t],\ \varnothing,\ 1$$

   where $l \notin \texttt{dom}(\sigma_i)$.

2. We can show that

   $$\varnothing,\ \sigma_c,\ \sigma_i,\ \beta \rightsquigarrow \sigma_c,\ \beta,\ 0$$

3. We trivially show that $0 \leq 0$

4. This goal is trivial since $\texttt{dom}(\beta) \setminus \texttt{dom}(\beta) = \emptyset$ and $\texttt{dom}'(\beta) \setminus \texttt{dom}'(\beta) = \emptyset$

5. We have to show that

   $$(v_i,\ v_c,\ l,\ (\sigma_i[l \mapsto v_t],\ \sigma_c,\ \beta,\ j)) \in \mathcal{V}_A[\![(\varphi A)^{\mathbb{C}}]\!]$$

   Which follows easily from eq. (A) and lemma 15.

- $$\dfrac{\Delta; \Phi_A \vdash^{\kappa'}_{\mathbb{S}} A \sqsubseteq A' \hookrightarrow \underline{\mathbf{g}} \qquad \Delta; \Phi \models \kappa \doteq \kappa'\ \hat{+}\ 2}{\Delta; \Phi \vdash^{\kappa}_{\mathbb{S}} (A)^{\mathbb{C}} \sqsubseteq (A')^{\mathbb{C}} \hookrightarrow \underline{\lambda} e.\texttt{let}\ l\ =\ \texttt{ref}\ \underline{\mathbf{g}}(!e)\ \texttt{in}\ \texttt{let}\ ()\ =\ \texttt{push}([l],\ \lambda().\,\texttt{ref}\ \underline{\mathbf{g}}(!e))\ \texttt{in}\ l}\ \textbf{C2-}\mathbb{S}$$

From the hypotheses we know

$$(v_i,\ v_c,\ l_t,\ (\sigma_i,\ \sigma_c,\ \beta,\ j)) \in \mathcal{V}[\![(\varphi A)^{\mathbb{C}}]\!]$$

thus we can derive

$$(v_i,\ \sigma_i(l_t),\ (\sigma_i,\ j)) \in \mathcal{V}_A(\!|\varphi A|\!)_{L_1} \tag{A1}$$

and

$$\forall\ m, (v_c,\ v_c,\ (\sigma_i + \sigma_c(\beta(l_t)),\ m)) \in \mathcal{V}_A(\!|\varphi A|\!)_{L_2} \tag{A2}$$

We can apply lemma 28 to eq. (A1) and obtain v_{t1}

$$\underline{\mathbf{g}}(\sigma_i(l_t)),\ \sigma_i \Downarrow^{L_1} v_{t1},\ \sigma_f,\ \varnothing,\ c_1$$

and

$$(v_i,\ v_{t1},\ (\sigma_f,\ j)) \in \mathcal{V}_A(\!|\varphi A'|\!)_{L_1}$$

From the above we can show that

$$\begin{gathered}\texttt{let}\ l\ =\ \texttt{ref}\ \underline{\mathbf{g}}(!l_t)\ \texttt{in}\ \texttt{let}\ ()\ =\ \texttt{push}([l],\ \lambda().\,\texttt{ref}\ \underline{\mathbf{g}}(!l_t))\ \texttt{in}\ l,\ \sigma_i \\ \Downarrow^{L_1} \\ l,\ \sigma_f[l \mapsto v_{t1}],\ ([l],\ \lambda().\,\texttt{ref}\ \underline{\mathbf{g}}(!l_t)),\ c_1 + 4\end{gathered}$$

and

$$(v_i,\ l,\ (\sigma_f[l \mapsto v_{t1}],\ j)) \in \mathcal{V}_A(\!|\varphi A'|\!)_{L_1} \tag{A}$$

Similarly we can apply lemma 28 to eq. (A2) and obtain v_{t2}

$$\underline{\mathbf{g}}(\sigma_c + \sigma_i(\beta(l_t))),\ \sigma_c + \sigma_i \Downarrow^{L_2} v_{t2},\ \sigma'_f + \sigma_i,\ \varnothing,\ c_2$$

$$c_2 \leq \varphi\kappa$$

and

$$\forall\ m, (v_c,\ v_{t2},\ (\sigma'_f,\ m)) \in \mathcal{V}_A(\!|\varphi A'|\!)_{L_1}$$

From the above we can show that

$$([l],\ \lambda().\,\texttt{ref}\ \underline{\mathbf{g}}(!l_t)),\ \sigma_c,\ \sigma_f,\ \beta \rightsquigarrow \sigma'_f[l' \mapsto v_{t2}],\ \beta[l \mapsto l'],\ c_2 + 2$$

and

$$\forall\ m, (v_c,\ l',\ (\sigma'_f[l' \mapsto v_{t2}],\ m)) \in \mathcal{V}_A(\!|\varphi A'|\!)_{L_1} \tag{B}$$

We can easily show that $c_2 + 2 \leq \varphi\kappa$. Furthermore from it follows that

$$(v_i,\ v_c,\ l,\ (\sigma_f[l \mapsto v_{t1}],\ \sigma'_f[l' \mapsto v_{t2}],\ [l \mapsto l'],\ j)) \in \mathcal{V}[\![(\varphi A')^{\mathbb{C}}]\!]$$

and since $l \notin \mathtt{dom}(\beta)$ and $l' \notin \mathtt{dom}'(\beta)$ we can derive

$$(v_i,\ v_c,\ l,\ (\sigma_f[l \mapsto v_{t1}],\ \sigma'_f[l' \mapsto v_{t2}],\ \beta[l \mapsto l'],\ j)) \in \mathcal{V}[\![(\varphi A')^{\mathbb{C}}]\!]$$

that proves the goal.

- $$\frac{\Delta;\Phi \vdash^{\kappa_1}_{\epsilon} \tau_1 \sqsubseteq \tau_2 \hookrightarrow \underline{\mathbf{g_1}} \qquad \Delta;\Phi \vdash^{\kappa_2}_{\epsilon} \tau_2 \sqsubseteq \tau_3 \hookrightarrow \underline{\mathbf{g_2}} \qquad \Delta;\Phi \models \kappa \doteq \kappa_1 \mathrel{\hat{+}} \kappa_2}{\Delta;\Phi \vdash^{\kappa}_{\epsilon} \tau_1 \sqsubseteq \tau_3 \hookrightarrow \underline{\lambda} e.\underline{\mathbf{g_2}}(\underline{\mathbf{g_1}}(e))}\,\textbf{tran}$$

This case follows easily from the induction hypothesis applied on the two premises.

- $$\frac{\Delta;\Phi \wedge C \vdash^{\kappa_1}_{\epsilon} \eta \hookrightarrow \underline{\mathbf{g_1}} \qquad \Delta;\Phi \wedge \dot{\neg} C \vdash^{\kappa_2}_{\epsilon} \eta \hookrightarrow \underline{\mathbf{g_2}}}{\Delta;\Phi \vdash^{0}_{\epsilon} \eta \hookrightarrow \underline{\lambda} e.\bot}\,\textbf{split}(\dagger)$$

This case follows by contradiction.

- $$\frac{\Delta;\Phi \wedge C' \models C \qquad \Delta;\Phi \vdash^{\kappa}_{\epsilon} \tau \sqsubseteq \tau' \hookrightarrow \underline{\mathbf{g}}}{\Delta;\Phi \vdash^{\kappa}_{\epsilon} C \to \tau \sqsubseteq C' \to \tau' \hookrightarrow \underline{\mathbf{g}}}\,\textbf{c-imp}$$

This case follows easily from the induction hypothesis.

- $$\frac{\Delta;\Phi \wedge C \models C' \qquad \Delta;\Phi \vdash^{\kappa}_{\epsilon} \tau \sqsubseteq \tau' \hookrightarrow \underline{\mathbf{g}}}{\Delta;\Phi \vdash^{\kappa}_{\epsilon} C \wedge \tau \sqsubseteq C' \wedge \tau' \hookrightarrow \underline{\mathbf{g}}}\,\textbf{c-and}$$

This case follows easily from the induction hypothesis.

□

Theorem 30 (Fundamental theorem - Unary interpretation)
Assume that the following hold:

$$\Delta;\Phi;\Gamma \vdash^{\kappa}_{\epsilon} e : \tau \hookrightarrow \ulcorner e \urcorner$$

$$\varphi \in \mathcal{D}[\![\Delta]\!]$$

$$(\theta_s,\ \theta_t,\ (\sigma,\ m)) \in \mathcal{G}(\!|\varphi\Gamma|\!)_L$$

$$\models \varphi\Phi$$

$$\varphi\epsilon = \mathbb{C}$$

Then

$$(\theta_s e,\ \theta_t \ulcorner e \urcorner,\ (\sigma,\ m)) \in \mathcal{E}(\!|\varphi\tau|\!)^{\varphi\kappa}_{L}$$

Proof. □

Theorem 31 (Fundamental theorem - Binary interpretation)
Assume that the following hold:

$$\Delta; \Phi; \Gamma \vdash_{\epsilon}^{\kappa} e : \tau \hookrightarrow \ulcorner e \urcorner$$

$$\varphi \in \mathcal{D}[\![\Delta]\!]$$

$$(\theta_i,\ \theta_c,\ \theta_t,\ (\sigma_i,\ \sigma_c,\ \beta,\ m)) \in \mathcal{G}[\![\varphi\Gamma]\!]$$

.

$$\varphi\epsilon = \mathbb{S}$$

$$\models \varphi\Phi$$

$$\mathtt{dom}(\beta) \subseteq \mathtt{dom}(\sigma_i) \wedge \mathtt{dom}'(\beta) \subseteq \mathtt{dom}(\sigma_c)$$

Then

$$(\theta_i e,\ \theta_c e,\ \theta_t \ulcorner e \urcorner,\ (\sigma_i,\ \sigma_c,\ \beta,\ m)) \in \mathcal{E}[\![\varphi\tau]\!]^{\varphi\kappa}$$

Proof. We proceed by induction on the typing derivation of e. For each of the following cases we pick φ, θ_i, θ_c, θ_t, σ_i, σ_c, β and m such that :

- $\varphi \in \mathcal{D}[\![\Delta]\!]$
- $\models \varphi\Phi$
- $\varphi\epsilon = \mathbb{S}$
- $(\theta_i,\ \theta_c,\ \theta_t,\ (\sigma_i,\ \sigma_c,\ \beta,\ m)) \in \mathcal{G}[\![\varphi\Gamma]\!]$

- $$\frac{\Delta; \Phi; \Gamma, f : (\tau_1 \xrightarrow{\delta(\kappa)} \tau_2)^{\mathbb{S}}, x : \tau_1 \vdash_{\delta}^{\kappa} e : \tau_2 \hookrightarrow \ulcorner e \urcorner}{\Delta; \Phi; \Gamma \vdash_{\epsilon}^{0} \mathtt{fix}\ f(x).\, e : (\tau_1 \xrightarrow{\delta(\kappa)} \tau_2)^{\mathbb{S}} \hookrightarrow \mathtt{fix}\ f(x).\, \ulcorner e \urcorner}\ \textbf{fix1}$$

We pick arbitrary $j_i < m$, v_i, $\sigma_i' \sqsupseteq^{L_1} \sigma_i$, $\sigma_c' \sqsupseteq^{L_2} \sigma_c$, $\beta' \geq \beta$ such that

$$\theta_i(\mathtt{fix}\ f(x).\, e) \Downarrow v_i,\ j_i$$

$$\mathtt{dom}(\beta') \setminus \mathtt{dom}(\beta) \subseteq \mathtt{dom}(\sigma_i') \setminus \mathtt{dom}(\sigma_i)$$

and

$$\mathtt{dom}'(\beta') \setminus \mathtt{dom}'(\beta) \subseteq \mathtt{dom}(\sigma_c') \setminus \mathtt{dom}(\sigma_c)$$

By inversion of the evaluation relation we derive that $j_i = 0$ and $v_i = \theta_i(\mathtt{fix}\ f(x).\, e)$

The goals are proved as follows:

1. We can easily derive that
$$\theta_c(\mathtt{fix}\ f(x).\, e) \Downarrow \theta_c(\mathtt{fix}\ f(x).\, e),\ 0$$

2. Similarly, we can show
$$\theta_t(\mathtt{fix}\ f(x).\, \ulcorner e \urcorner),\ \sigma_i' \Downarrow^{L_1} \theta_t(\mathtt{fix}\ f(x).\, \ulcorner e \urcorner),\ \sigma_i',\ \varnothing,\ 0$$

3. From the definition of the change propagation algorithm we can easily obtain that:

$$\varnothing,\ \sigma'_c,\ \sigma_i,\ \beta' \rightsquigarrow \sigma'_c,\ \beta',\ 0$$

4. It is obvious that

$$0 \leq 0$$

5. we can trivially show that $(\sigma_i,\ \sigma_c,\ \beta,\ j) \geq (\sigma_i,\ \sigma_c,\ \beta,\ j)$
6. Finally, we need to show

$$(\theta_i(\mathtt{fix}\ f(x).\, e),\ \theta_c(\mathtt{fix}\ f(x).\, e),\ \theta_t(\mathtt{fix}\ f(x).\, \ulcorner e \urcorner),\ (\sigma'_i,\ \sigma'_c,\ \beta',\ m)) \in \mathcal{V}[\![(\varphi\tau_1 \xrightarrow{\delta(\varphi\kappa)} \varphi\tau_2)^{\mathbb{S}}]\!]$$

We proceed by case analysis.

▶ $\delta = \mathbb{S}$

By the induction hypothesis we obtain that

$$\begin{aligned}\forall\ (\theta_i,\ \theta_c,\ \theta_t,\ (\sigma_i,\ \sigma_c,\ \beta,\ m)) \in \mathcal{G}[\![\varphi\Gamma, f : (\varphi\tau_1 \xrightarrow{\mathbb{S}(\varphi\kappa)} \varphi\tau_2)^{\mathbb{S}}, x : \varphi\tau_1]\!],\\ (\theta_i e_1,\ \theta_c e_1,\ \theta_t \ulcorner e_1 \urcorner,\ (\sigma_i,\ \sigma_c,\ \beta,\ m)) \in \mathcal{E}[\![\varphi\tau_2]\!]^{\varphi\kappa}\end{aligned} \tag{A}$$

We proceed by induction on m.

⋆ $m = 0$

The result holds vacuously from the definition of the logical relation.

⋆ $m = m' + 1$

From the induction hypothesis we obtain

$$(\theta_i(\mathtt{fix}\ f(x).\, e),\ \theta_c(\mathtt{fix}\ f(x).\, e),\ \theta_t(\mathtt{fix}\ f(x).\, \ulcorner e \urcorner),\ (\sigma'_i,\ \sigma'_c,\ \beta',\ m')) \in \mathcal{V}[\![(\varphi\tau_1 \xrightarrow{\mathbb{S}(\varphi(\kappa))} \varphi\tau_2)^{\mathbb{S}}]\!] \tag{B}$$

We pick arbitrary $W > (\sigma'_i,\ \sigma'_c,\ \beta',\ m'+1)$ such that

$$(v_i,\ v_c,\ v_t,\ W) \in \mathcal{V}[\![\varphi\tau_1]\!] \tag{1}$$

We can apply lemma 15 to eq. (B) and derive

$$(\theta_i(\mathtt{fix}\ f(x).\, e),\ \theta_c(\mathtt{fix}\ f(x).\, e),\ \theta_t(\mathtt{fix}\ f(x).\, \ulcorner e \urcorner),\ W) \in \mathcal{V}[\![(\varphi\tau_1 \xrightarrow{\mathbb{S}(\varphi\kappa)} \varphi\tau_2)^{\mathbb{S}}]\!] \tag{2}$$

From the hypotheses we know that

$$(\theta_i,\ \theta_c,\ \theta_t,\ (\sigma'_i,\ \sigma'_c,\ \beta',\ m)) \in \mathcal{G}[\![\varphi\Gamma]\!]$$

Similarly, we can use lemma 15 to derive

$$(\theta_i,\ \theta_c,\ \theta_t,\ W) \in \mathcal{G}[\![\varphi\Gamma]\!]$$

From the above and eq. (1), eq. (2) we can easily show that

$$(\theta_i[x \mapsto v_i, f \mapsto \texttt{fix}\ f(x).\, \theta_i e_i],\ \theta_c[x \mapsto v_c, f \mapsto \texttt{fix}\ f(x).\, \theta_c e_c],\ \theta_t[x \mapsto v_t, f \mapsto \texttt{fix}\ f(x).\, \theta_t e_t],\ W)$$
$$\in \mathcal{G}[\![\varphi\Gamma, f : (\varphi\tau_1 \xrightarrow{\mathbb{S}(\varphi\kappa)} \varphi\tau_2)^{\mathbb{S}}]\!]$$

We can instantiate eq. (A) with the above and derive that

$$(\theta_i[x \mapsto v_i, f \mapsto \texttt{fix}\ f(x).\, \theta_i e]e,\ \theta_c[x \mapsto v_c, f \mapsto \texttt{fix}\ f(x).\, \theta_c e]e,\ \theta_t[x \mapsto v_t, f \mapsto \texttt{fix}\ f(x).\, \theta_t \ulcorner e \urcorner]\ulcorner e \urcorner,\ W)$$
$$\in \mathcal{E}[\![\varphi\tau_2]\!]^{\varphi\kappa}$$

which proves the goal.

▶ $\delta = \mathbb{C}$

By the definition of the logical relation it suffices to show

$$(\theta_i(\texttt{fix}\ f(x).\, e),\ \theta_t(\texttt{fix}\ f(x).\, \ulcorner e \urcorner),\ (\sigma_i',\ m)) \in \mathcal{V}_A(\!|\varphi\tau_1 \xrightarrow{\mathbb{C}(\varphi\kappa)} \varphi\tau_2|\!)_{L_1}$$

and

$$\forall\ m,\ (\theta_c(\texttt{fix}\ f(x).\, e),\ \beta'(\theta_t(\texttt{fix}\ f(x).\, \ulcorner e \urcorner)),\ (\sigma_c' + \sigma_i',\ m)) \in \mathcal{V}_A(\!|\varphi\tau_1 \xrightarrow{\mathbb{C}(\varphi\kappa)} \varphi\tau_2|\!)_{L_2}$$

To show the first goal we will proceed by induction on m.

⋆ $m = 0$

The result holds vacuously from the definition of the logical relation.

⋆ $m = m' + 1$

From the induction hypothesis we obtain

$$(\theta_i(\texttt{fix}\ f(x).\, e),\ \theta_t(\texttt{fix}\ f(x).\, \ulcorner e \urcorner),\ (\sigma_i',\ m')) \in \mathcal{V}_A(\!|\varphi\tau_1 \xrightarrow{\mathbb{C}(\varphi\kappa)} \varphi\tau_2|\!)_{L_1} \tag{A}$$

We pick arbitrary $j < m' + 1$, $\sigma_i'' \sqsupseteq^{L_1} \sigma_i'$, v_i, and v_t such that

$$(v_i,\ v_t,\ (\sigma_i'',\ j)) \in \mathcal{V}(\!|\varphi\tau_1|\!)_{L_1} \tag{1}$$

Since $j \leq m'$ and $\sigma_i'' \sqsupseteq^{L_1} \sigma_i'$, we can apply **??** and **??** to eq. (A) and derive

$$(\theta_i(\texttt{fix}\ f(x).\, e),\ \theta_t(\texttt{fix}\ f(x).\, \ulcorner e \urcorner),\ (\sigma_i'',\ j)) \in \mathcal{V}_A(\!|\varphi\tau_1 \xrightarrow{\mathbb{C}(\varphi\kappa)} \varphi\tau_2|\!)_{L_1} \tag{2}$$

From the hypotheses we know that

$$(\theta_i,\ \theta_c,\ \theta_t,\ (\sigma_i,\ \sigma_c,\ \beta,\ m)) \in \mathcal{G}[\![\varphi\Gamma]\!]$$

We can apply lemma 16 and lemma 15 to derive

$$(\theta_i,\ \theta_t,\ (\sigma_i'',\ j)) \in \mathcal{G}\llparenthesis \varphi\Gamma \rrparenthesis_{L_1}$$

From the above, eq. (1) and eq. (2) we can easily show that

$$(\theta_i[x \mapsto v_i, f \mapsto \texttt{fix}\, f(x).\, \theta_i e_i],\ \theta_t[x \mapsto v_t, f \mapsto \texttt{fix}\, f(x).\, \theta_t e_t],\ (\sigma_i'',\ j)) \in \mathcal{G}\llparenthesis \varphi\Gamma, f : (\varphi\tau_1 \xrightarrow{\mathbb{S}(\varphi\kappa)} \varphi\tau_2)^{\mathbb{S}} \rrparenthesis_{L_1}$$

We can now apply theorem 30 to the premise of the typing derivation and instantiate the result with the above statement in order to derive that

$$(\theta_i[x \mapsto v_i, f \mapsto \texttt{fix}\ f(x).\, \theta_i e]e,\ \theta_t[x \mapsto v_t, f \mapsto \texttt{fix}\ f(x).\, \theta_t \ulcorner e \urcorner]\ulcorner e \urcorner,\ (\sigma_i'',\ j)) \in \mathcal{E}\llparenthesis \varphi\tau_2 \rrparenthesis_{L_1}^{\varphi\kappa}$$

that proves the goal.

To prove the second goal we pick an arbitrary m. Using lemma 14 we can show that $\beta'(\ulcorner e \urcorner) = \ulcorner e \urcorner$. Consequently, it suffices to show that

$$\forall\ m,\ (\theta_c(\texttt{fix}\ f(x).\, e),\ (\beta' \cdot \theta_t)(\texttt{fix}\ f(x).\, \ulcorner e \urcorner),\ (\sigma_c' + \sigma_i',\ m)) \in \mathcal{V}_A\llparenthesis \varphi\tau_1 \xrightarrow{\mathbb{C}(\varphi\kappa)} \varphi\tau_2 \rrparenthesis_{L_2}$$

We proceed by induction on m.

⋆ $m = 0$

The result holds vacuously from the definition of the logical relation.

⋆ $m = m' + 1$

From the induction hypothesis we obtain

$$(\theta_c(\texttt{fix}\ f(x).\, e),\ (\beta' \cdot \theta_t)(\texttt{fix}\ f(x).\, \ulcorner e \urcorner),\ (\sigma_c',\ m')) \in \mathcal{V}_A\llparenthesis \varphi\tau_1 \xrightarrow{\mathbb{C}(\varphi\kappa)} \varphi\tau_2 \rrparenthesis_{L_2} \qquad \text{(B)}$$

We pick arbitrary $j < m' + 1$, $\sigma'' \sqsupseteq^{L_1 \cup L_2} \sigma_c' + \sigma_i'$, v_c, and v_t such that

$$(v_c,\ v_t,\ (\sigma'',\ j)) \in \mathcal{V}\llparenthesis \varphi\tau_1 \rrparenthesis_{L_2} \qquad (3)$$

Since $j \leq m'$ and $\sigma'' \sqsupseteq^{L_1 \cup L_2} \sigma_c'$, we can apply lemma 15 to eq. (B) and derive

$$(\theta_c(\texttt{fix}\ f(x).\, e),\ (\beta' \cdot \theta_t)(\texttt{fix}\ f(x).\, \ulcorner e \urcorner),\ (\sigma'',\ j)) \in \mathcal{V}_A\llparenthesis \varphi\tau_1 \xrightarrow{\mathbb{C}(\varphi\kappa)} \varphi\tau_2 \rrparenthesis_{L_2} \qquad (4)$$

From the hypotheses we know that

$$(\theta_i,\ \theta_c,\ \theta_t,\ (\sigma_i,\ \sigma_c,\ \beta,\ m)) \in \mathcal{G}\llbracket \varphi\Gamma \rrbracket$$

We can apply lemma 15 and lemma 16 and derive

$$(\theta_c,\ \beta' \cdot \theta_t,\ (\sigma'',\ j)) \in \mathcal{G}\llparenthesis \varphi\Gamma \rrparenthesis_{L_2}$$

From the above and eq. (3), eq. (4) we can derive that

$$(\theta_c[x \mapsto v_i, f \mapsto \texttt{fix}\ f(x).\, \theta_c e_c],\ (\beta' \cdot \theta_t)[x \mapsto v_t, f \mapsto \texttt{fix}\ f(x).\, (\beta' \cdot \theta_t)e_t],\ (\sigma'',\ j)) \\ \in \mathcal{G}(\!|\varphi\Gamma, f : (\varphi\tau_1 \xrightarrow{\mathbb{S}(\varphi\kappa)} \varphi\tau_2)^{\mathbb{S}}|\!)_{L_2}$$

We can now apply theorem 30 to the premise of the typing derivation and instantiate the result with the above statement in order to derive that

$$(\theta_i[x \mapsto v_i, f \mapsto \texttt{fix}\ f(x).\, \theta_i e]e,\ (\beta'\cdot\theta_t)[x \mapsto v_t, f \mapsto \texttt{fix}\ f(x).\, (\beta'\cdot\theta_t)\ulcorner e\urcorner]\ulcorner e\urcorner,\ (\sigma''_c,\ j)) \in \mathcal{E}(\!|\varphi\tau_2|\!)^{\varphi\kappa}_{L_2}$$

which proves the goal.

- $$\frac{\begin{array}{c}\Delta;\Phi;\Gamma \vdash^{\kappa_1}_{\mathbb{S}} e_1 : (\tau_1 \xrightarrow{\mathbb{S}(\kappa')} \tau_2)^{\mu} \hookrightarrow \ulcorner e_1\urcorner \\ \Delta;\Phi;\Gamma \vdash^{\kappa_2}_{\mathbb{S}} e_2 : \tau_1 \hookrightarrow \ulcorner e_2\urcorner \qquad \mu \leq \mathbb{S} \qquad \Delta;\Phi \models \kappa \doteq \kappa' \mathbin{\hat{+}} \kappa_1 \mathbin{\hat{+}} \kappa_2\end{array}}{\Delta;\Phi;\Gamma \vdash^{\kappa}_{\mathbb{S}} e_1\ e_2 : \tau_2 \hookrightarrow \ulcorner e_1\urcorner\, \ulcorner e_2\urcorner}\ \textbf{app1}$$

By the induction hypothesis applied on the premises we get:

$$(\theta_i e_1,\ \theta_c e_1,\ \theta_t\ulcorner e_1\urcorner,\ (\sigma_i,\ \sigma_c,\ \beta,\ m)) \in \mathcal{E}[\![(\varphi\tau_1 \xrightarrow{\mathbb{S}(\varphi\kappa')} \varphi\tau_2)^{\mathbb{S}}]\!]^{\varphi\kappa_1} \tag{A}$$

and

$$(\theta_i e_2,\ \theta_c e_2,\ \theta_t\ulcorner e_2\urcorner,\ (\sigma_i,\ \sigma_c,\ \beta,\ m)) \in \mathcal{E}[\![\varphi\tau_1]\!]^{\varphi\kappa_2} \tag{B}$$

We pick arbitrary $j_i < m$, v_i, $\sigma'_i \sqsupseteq^{L_1} \sigma_i$, $\sigma'_c \sqsupseteq^{L_2} \sigma_c$, $\beta' \geq \beta$ such that

$$\theta_i(e_1\ e_2) \Downarrow v_i,\ j_i$$

$$\texttt{dom}(\beta') \setminus \texttt{dom}(\beta) \subseteq \texttt{dom}(\sigma'_i) \setminus \texttt{dom}(\sigma_i)$$

and

$$\texttt{dom}'(\beta') \setminus \texttt{dom}'(\beta) \subseteq \texttt{dom}(\sigma'_c) \setminus \texttt{dom}(\sigma_c)$$

By inversion of the evaluation relation we derive the following:

$$\theta_i e_1 \Downarrow \texttt{fix}\ f(x).\, e'_i,\ j_1 \tag{1}$$

$$\theta_i e_2 \Downarrow v'_i,\ j_2 \tag{2}$$

$$e'_i[x/v'_i][f/\texttt{fix}\ f(x).\, e'_i] \Downarrow v_i,\ j_3 \tag{3}$$

$$j_i = j_1 + j_2 + j_3 + 1$$

for some e'_i, v'_i, j_1, j_2 and j_3

We instantiate eq. (A) with j_1 (note that $j_1 < m$), σ'_i, σ'_c, β' (note that $\sigma'_i \sqsupseteq^{L_1} \sigma_i$, $\sigma'_c \sqsupseteq^{L_2} \sigma_c$, $\beta' \geq \beta$ and the required conditions for the bijections hold), and eq. (1) and we obtain v_{c1}, j_1', v_{t1}, σ_1, D_1, c_1, σ_1', β_1, c_1' such that:

$$\theta_c e_1 \Downarrow v_{c1},\ j_1' \tag{A1}$$

$$\theta_t \ulcorner e_1 \urcorner,\ \sigma_i \Downarrow^{L_1} v_{t1},\ \sigma_1,\ D_1,\ c_1 \tag{A2}$$

$$D_1,\ \sigma_c',\ \sigma_1,\ \beta' \rightsquigarrow \sigma_1',\ \beta_1,\ c_1' \tag{A3}$$

$$c_1' \leq \varphi\kappa_1 \tag{A4}$$

$$\mathtt{dom}(\beta_1) \setminus \mathtt{dom}(\beta') \subseteq \mathtt{dom}(\sigma_1) \setminus \mathtt{dom}(\sigma_i') \wedge \mathtt{dom}'(\beta_1) \setminus \mathtt{dom}'(\beta') \subseteq \mathtt{dom}(\sigma_1') \setminus \mathtt{dom}(\sigma_c') \tag{A5}$$

and

$$(\mathtt{fix}\ f(x).\, e_i',\ v_{c1},\ v_{t1},\ (\sigma_1,\ \sigma_1',\ \beta_1,\ m - j_1)) \in \mathcal{V}\llbracket (\varphi\tau_1 \xrightarrow{\mathbb{S}(\varphi\kappa')} \varphi\tau_2)^{\mathbb{S}} \rrbracket$$

By unfolding the last statement we can derive that

$$(\mathtt{fix}\ f(x).\, e_i',\ v_{c1},\ v_{t1},\ (\sigma_1,\ \sigma_1',\ \beta_1,\ m - j_1)) \in \mathcal{V}_A\llbracket \varphi\tau_1 \xrightarrow{\mathbb{S}(\varphi\kappa')} \varphi\tau_2 \rrbracket \tag{A6}$$

and that $v_{c1} = \mathtt{fix}\ f(x).\, e_c'$, $v_{t1} = \mathtt{fix}\ f(x).\, e_t'$ for some e_c', e_t'.

We now instantiate eq. (B) with j_2 (note that $j_2 < m$) σ_1, σ_1', β_1 (note that $\sigma_1 \sqsupseteq^{L_1} \sigma_i$, $\sigma_1' \sqsupseteq^{L_2} \sigma_c$ and $\beta_1 \geq \beta$) eq. (A5) and eq. (2), and we obtain v_c', j_2', v_t', σ_2, D_2, c_2, σ_2', β_2, c_2' such that:

$$\theta_c e_2 \Downarrow v_c',\ j_2' \tag{B1}$$

$$\theta_t \ulcorner e_2 \urcorner,\ \sigma_1 \Downarrow^{L_1} v_t',\ \sigma_2,\ D_2,\ c_2 \tag{B2}$$

$$D_2,\ \sigma_1',\ \sigma_2,\ \beta_1 \rightsquigarrow \sigma_2',\ \beta_2,\ c_2' \tag{B3}$$

$$c_2' \leq \varphi\kappa_2 \tag{B4}$$

$$\mathtt{dom}(\beta_2) \setminus \mathtt{dom}(\beta_1) \subseteq \mathtt{dom}(\sigma_2) \setminus \mathtt{dom}(\sigma_1) \wedge \mathtt{dom}'(\beta_2) \setminus \mathtt{dom}'(\beta_1) \subseteq \mathtt{dom}(\sigma_2') \setminus \mathtt{dom}(\sigma_1') \tag{B5}$$

and

$$(v_i',\ v_c',\ v_t',\ (\sigma_2,\ \sigma_2',\ \beta_2,\ m - j_2)) \in \mathcal{V}\llbracket \varphi\tau_1 \rrbracket$$

We apply lemma 15 at the last statement and we obtain

$$(v_i',\ v_c',\ v_t',\ (\sigma_2,\ \sigma_2',\ \beta_2,\ m - j_1 - j_2 - 1)) \in \mathcal{V}\llbracket \varphi\tau_1 \rrbracket \tag{B6}$$

We instantiate eq. (A6) with σ_2, σ_2', β_2, eq. (B5) and eq. (B6) and we derive

$$(e_i'[x/v_i'][f/\ldots],\ e_c'[x/v_c'][f/\ldots],\ e_t'[x/v_t'][f/\ldots],\ (\sigma_2,\ \sigma_2',\ \beta_2,\ m - j_1 - j_2 - 1)) \in \mathcal{E}\llbracket \varphi\tau_2 \rrbracket^{\varphi\kappa'} \tag{C}$$

We now instantiate eq. (C) with eq. (B5), j_3 (note that $j_3 < m - j_1 - j_2 - 1$) σ_2, σ_2', β_2 (note that $\sigma_2 \sqsupseteq^{L_1} \sigma_1$, $\sigma_2' \sqsupseteq^{L_2} \sigma_1'$ and the conditions for the bijection hold trivially) and eq. (3), and and we obtain v_c, j_3', v_t, σ_3,

D_3, c_3, σ'_3, β_3, c'_3 such that:

$$e'_c[x/v'_c][f/\texttt{fix}\ f(x).\, e'_c] \Downarrow v_c,\ j'_3 \tag{C1}$$

$$e'_t[x/v'_t][f/\texttt{fix}\ f(x).\, e'_t],\ \sigma_2 \Downarrow^{L_1} v_t,\ \sigma_3,\ D_3,\ c_3 \tag{C2}$$

$$D_3,\ \sigma'_2,\ \sigma_3,\ \beta_2 \rightsquigarrow \sigma'_3,\ \beta_3,\ c'_3 \tag{C3}$$

$$c' \leq \varphi\kappa' \tag{C4}$$

$$\texttt{dom}(\beta_3) \setminus \texttt{dom}(\beta_2) \subseteq \texttt{dom}(\sigma_3) \setminus \texttt{dom}(\sigma_2) \wedge \texttt{dom}'(\beta_3) \setminus \texttt{dom}'(\beta_2) \subseteq \texttt{dom}(\sigma'_3) \setminus \texttt{dom}(\sigma'_2) \tag{C5}$$

and

$$(v_i,\ v_c,\ v_t,\ (\sigma_3,\ \sigma'_3,\ \beta_3,\ m - j_i)) \in \mathcal{V}[\![\varphi\tau_2]\!] \tag{C6}$$

We can now show the goals:

1. From eq. (A1), eq. (B1) and eq. (C1) we can derive that

$$\theta_c(e_1\ e_2) \Downarrow v_c,\ j'_1 + j'_2 + j'_3 + 1$$

2. From eq. (A2), eq. (B2) and eq. (C2) we can derive that

$$\theta_t(\ulcorner e_1 \urcorner\ \ulcorner e_2 \urcorner),\ \sigma'_i \Downarrow^{L_1} v_t,\ \sigma_3,\ D_1 + D_2 + D_3,\ c_1 + c_2 + c_3 + 1$$

3. From eq. (A3), eq. (B3) and eq. (C3) using lemma 13 and lemma 11 we can derive that

$$D_1 + D_2 + D_3,\ \sigma'_i,\ \sigma_3,\ \beta' \rightsquigarrow \sigma'_3,\ \beta_3,\ c'_1 + c'_2 + c'_3$$

4. From eq. (A4), eq. (B4) and eq. (C4) we can derive that

$$c'_1 + c'_2 + c'_3 \leq \varphi(\kappa_1 + \kappa_2 + \kappa')$$

5. From eq. (A5), eq. (B5) and eq. (C5) we derive that

$$\texttt{dom}(\beta_3) \setminus \texttt{dom}(\beta') \subseteq \texttt{dom}(\sigma_3) \setminus \texttt{dom}(\sigma'_i) \wedge \texttt{dom}'(\beta_3) \setminus \texttt{dom}'(\beta') \subseteq \texttt{dom}(\sigma'_3) \setminus \texttt{dom}(\sigma'_c)$$

6. From eq. (C6) we derive that

$$(v_i,\ v_c,\ v_t,\ (\sigma_3,\ \sigma'_3,\ \beta_3,\ m - j_i)) \in \mathcal{V}[\![\varphi\tau_2]\!]$$

- $$\frac{\begin{array}{c}\Delta; \Phi; \Gamma \vdash^{\kappa_1}_{\mathbb{S}} e_1 : (\tau_1 \xrightarrow{\mathbb{C}(\kappa')} \tau_2)^{\mathbb{C}} \hookrightarrow \ulcorner e_1 \urcorner \\ \Delta; \Phi; \Gamma \vdash^{\kappa_2}_{\mathbb{S}} e_2 : \tau_1 \hookrightarrow \ulcorner e_2 \urcorner \qquad \mathbb{C} \trianglelefteq \tau_2 \qquad \Delta; \Phi \models \kappa \doteq \kappa' \mathrel{\hat{+}} \kappa_1 \mathrel{\hat{+}} \kappa_2 \mathrel{\hat{+}} \texttt{cost}_{\texttt{deepref}}(\tau_2) \mathrel{\hat{+}} 3\end{array}}{\Delta; \Phi; \Gamma \vdash^{\kappa}_{\mathbb{S}} e_1\ e_2 : \tau_2 \hookrightarrow \begin{array}{l}\texttt{let}\ l\ =\ \ulcorner e_1 \urcorner\ \texttt{in let}\ x\ =\ \ulcorner e_2 \urcorner\ \texttt{in let}\ r\ =\ \underline{\textbf{deep}}(!l\ x,\ \tau_2)\ \texttt{in} \\ \texttt{let}\ ()\ =\ \texttt{push}(\texttt{snd}\ r,\ \lambda().\, \underline{\textbf{deep}'}(!l\ x,\ \tau_2))\ \texttt{in fst}\ r\end{array}}\ \textbf{app2}$$

By the induction hypothesis applied on the premises we get:

$$(\theta_i e_1,\ \theta_c e_1,\ \theta_t\ulcorner e_1\urcorner,\ (\sigma_i,\ \sigma_c,\ \beta,\ m)) \in \mathcal{E}[\![(\varphi\tau_1 \xrightarrow{\mathbb{C}(\varphi\kappa')} \varphi\tau_2)^{\mathbb{C}}]\!]^{\varphi\kappa_1} \tag{A}$$

$$(\theta_i e_2,\ \theta_c e_2,\ \theta_t\ulcorner e_2\urcorner,\ (\sigma_i,\ \sigma_c,\ \beta,\ m)) \in \mathcal{E}[\![\varphi\tau_1]\!]^{\varphi\kappa_2} \tag{B}$$

Let $\mathtt{dom}(\beta) \subseteq \mathtt{dom}(\sigma_i)$ and $\mathtt{dom}'(\beta) \subseteq \mathtt{dom}(\sigma_c)$. We pick arbitrary $j_i < m$, v_i, $\sigma_i' \sqsupseteq^{L_1} \sigma_i$, $\sigma_c' \sqsupseteq^{L_2} \sigma_c$, $\beta' \geq \beta$ such that

$$\theta_i(e_1\ e_2) \Downarrow v_i,\ j_i$$

$$\mathtt{dom}(\beta') \setminus \mathtt{dom}(\beta) \subseteq \mathtt{dom}(\sigma_i') \setminus \mathtt{dom}(\sigma_i)$$

and

$$\mathtt{dom}'(\beta') \setminus \mathtt{dom}'(\beta) \subseteq \mathtt{dom}(\sigma_c') \setminus \mathtt{dom}(\sigma_c)$$

By inversion of the evaluation relation we derive the following:

$$\theta_i e_1 \Downarrow \mathtt{fix}\ f(x).\, e_i',\ j_1 \tag{1}$$

$$\theta_i e_2 \Downarrow v_i',\ j_2 \tag{2}$$

$$e_i'[x/v_i'][f/\mathtt{fix}\ f(x).\, e_i'] \Downarrow v_i,\ j_3 \tag{3}$$

$$j_i = j_1 + j_2 + j_3 + 1$$

for some e_i', v_i', j_1, j_2 and j_3.

We instantiate eq. (A) with j_1 (note that $j_1 < m$) σ_i', σ_c', β' (note that $\sigma_i' \sqsupseteq^{L_1} \sigma_i$, $\sigma_c' \sqsupseteq^{L_2} \sigma_c$, $\beta' \geq \beta$, and all the preconditions for the bijection hold) and eq. (1) and we obtain v_{c1}, j_1', l, σ_1, D_1, c_1, σ_1', β_1, c_1' such that:

$$\theta_c e_1 \Downarrow v_{c1},\ j_1' \tag{A1}$$

$$\theta_t\ulcorner e_1\urcorner,\ \sigma_i \Downarrow^{L_1} l,\ \sigma_1,\ D_1,\ c_1 \tag{A2}$$

$$D_1,\ \sigma_c',\ \sigma_1,\ \beta' \rightsquigarrow \sigma_1',\ \beta_1,\ c_1' \tag{A3}$$

$$c_1' \leq \varphi\kappa_1 \tag{A4}$$

$$\mathtt{dom}(\beta_1) \setminus \mathtt{dom}(\beta') \subseteq \mathtt{dom}(\sigma_1) \setminus \mathtt{dom}(\sigma_i') \wedge \mathtt{dom}'(\beta_1) \setminus \mathtt{dom}'(\beta') \subseteq \mathtt{dom}(\sigma_1') \setminus \mathtt{dom}(\sigma_c') \tag{A5}$$

and

$$(\mathtt{fix}\ f(x).\, e_i',\ v_{c1},\ l,\ (\sigma_1,\ \sigma_1',\ \beta_1,\ m - j_1)) \in \mathcal{V}[\![(\varphi\tau_1 \xrightarrow{\mathbb{C}(\varphi\kappa')} \varphi\tau_2)^{\mathbb{C}}]\!]$$

By unfolding the last statement we can derive that $l \in \mathtt{dom}(\sigma_1)$, $\beta_1(l) \in \mathtt{dom}(\sigma_1' + \sigma_1)$ and

$$(\mathtt{fix}\ f(x).\, e_i',\ \sigma_1(l),\ (\sigma_1,\ m - j_1)) \in \mathcal{V}_A(\!|\varphi\tau_1 \xrightarrow{\mathbb{C}(\varphi\kappa')} \varphi\tau_2|\!)_{L_1} \tag{A6.1}$$

$$\forall\ m,\ (v_{c1},\ \sigma'_1 + \sigma_1(\beta_1(l)),\ (\sigma'_1 + \sigma_1,\ m)) \in \mathcal{V}_A(\!|\varphi\tau_1 \xrightarrow{\mathbb{C}(\varphi\kappa')} \varphi\tau_2|\!)_{L_2} \tag{A6.2}$$

From eq. (A6.1) we can derive that $\sigma_1(l) = \texttt{fix}\ f(x).\, e'_{t1}$ for some e'_{t1}. By instantiating eq. (A6.2) with a concrete step index we can derive that $v_{c1} = \texttt{fix}\ f(x).\, e'_c$ and $\sigma'_1 + \sigma_1(\beta_1(l)) = \texttt{fix}\ f(x).\, e'_{t2}$ for some e'_c, e'_{t2}. We now instantiate eq. (B) with j_2 (note that $j_2 < m$) σ_1, σ'_1, β_1 (note that $\sigma_1 \sqsupseteq^{L_1} \sigma'_i$, $\sigma'_1 \sqsupseteq^{L_2} \sigma'_c$, $\beta_1 \geq \beta'$), eq. (2), and eq. (A6) and we obtain v'_c, j'_2, v'_t, σ_2, D_2, c_2, σ'_2, β_2, c'_2 such that:

$$\theta_c e_2 \Downarrow v'_c,\ j'_2 \tag{B1}$$

$$\theta_t \ulcorner e_2 \urcorner,\ \sigma_1 \Downarrow^{L_1} v'_t,\ \sigma_2,\ D_2,\ c_2 \tag{B2}$$

$$D_2,\ \sigma'_1,\ \beta_1 \rightsquigarrow \sigma'_2,\ \beta_2,\ c'_2 \tag{B3}$$

$$c'_2 \leq \varphi\kappa_2 \tag{B4}$$

$$\texttt{dom}(\beta_2) \setminus \texttt{dom}(\beta_1) \subseteq \texttt{dom}(\sigma_2) \setminus \texttt{dom}(\sigma_1) \wedge \texttt{dom}'(\beta_2) \setminus \texttt{dom}'(\beta_1) \subseteq \texttt{dom}(\sigma'_2) \setminus \texttt{dom}(\sigma'_1) \tag{B5}$$

and

$$(v'_i,\ v'_c,\ v'_t,\ (\sigma_2,\ \sigma'_2,\ \beta_2,\ m - j_2)) \in \mathcal{V}[\![\varphi\tau_1]\!]$$

We apply lemma 15 and lemma 16 at the last statement and we obtain

$$(v'_i,\ v'_t,\ (\sigma_2,\ m - j_1 - j_2 - 1)) \in \mathcal{V}(\!|\varphi\tau_1|\!)_{L_1} \tag{B6.1}$$

and

$$\forall\ m,\ (v'_c,\ \beta_2(v'_t),\ (\sigma'_2 + \sigma_2,\ m)) \in \mathcal{V}(\!|\varphi\tau_1|\!)_{L_2} \tag{B6.2}$$

We instantiate eq. (A6.1) with σ_2 (note that $\sigma_2 \sqsupseteq^{L_1} \sigma_1$) and and eq. (B6.1) and we derive

$$(e'_i[x/v'_i][f/\texttt{fix}\ f(x).\, e'_i],\ e'_{t1}[x/v'_t][f/\texttt{fix}\ f(x).\, e'_{t1}],\ (\sigma_2,\ m - j_1 - j_2 - 1)) \in \mathcal{E}(\!|\varphi\tau_2|\!)^{\varphi\kappa'}_{L_1} \tag{C}$$

Using eq. (A6.2) and eq. (B6.2), we can also derive

$$\forall\ m,\ (e'_c[x/v'_c][f/\texttt{fix}\ f(x).\, e'_c],\ e'_{t2}[x/\beta_2(v'_t)][f/\texttt{fix}\ f(x).\, e'_{t2}],\ (\sigma'_2 + \sigma_2,\ m)) \in \mathcal{E}(\!|\varphi\tau_2|\!)^{\varphi\kappa'}_{L_2} \tag{D}$$

From eq. (C) using the fact that the evaluation relation in the source language is deterministic and that $j_3 < m - j_1 - j_2 - 1$ we obtain v_t, σ_3, c_3 such that

$$e'_{t1}[x/v'_t][f/\texttt{fix}\ f(x).\, e'_{t1}],\ \sigma_2 \Downarrow^{L_1} v_{t1},\ \sigma_3,\ \varnothing,\ c_3$$

and consequently, since $\sigma_2(l) = \sigma_1(l) = \texttt{fix}\ f(x).\, e'_{t1}$, we obtain

$$!l\ v'_t,\ \sigma_2 \Downarrow^{L_1} v_{t1},\ \sigma_3,\ \varnothing,\ c_3 + 2 \tag{C1}$$

We also obtain

$$c_3 \leq \varphi\kappa' \tag{C2}$$

$$(v_i,\ v_{t1},\ (\sigma_3,\ m - j_i)) \in \mathcal{V}\llparenthesis\varphi\tau_2\rrparenthesis_{L_1} \tag{C3}$$

We now instantiate lemma 18 with eq. (D) and we derive that

$$e'_c[x/v'_c][f/\texttt{fix}\ f(x).\, e'_c] \Downarrow v_c,\ j'_3 \tag{D1}$$

$$e'_{t2}[x/\beta_2(v'_t)][f/\texttt{fix}\ f(x).\, e'_{t2}],\ \sigma'_2 + \sigma_2 \Downarrow^{L_2} v_{t2},\ \sigma'_3 + \sigma_2,\ \varnothing,\ 0c'_3$$

and consequently, since $\sigma'_2 + \sigma_2(\beta_2(l)) = \sigma'_1 + \sigma_1(\beta_1(l)) = \texttt{fix}\ f(x).\, e'_{t2}$ (note that $l \in \texttt{dom}(\sigma_1)$ thus $\beta_1(l) = \beta_2(l)$), we can derive

$$\beta_2(!l)\ \beta_2(v'_t),\ \sigma'_2 + \sigma_2 \Downarrow^{L_2} v_{t2},\ \sigma'_3 + \sigma_2,\ \varnothing,\ c'_3 + 2 \tag{D2}$$

We also derive

$$c'_3 \leq \varphi\kappa' \tag{D3}$$

$$\forall\ m,\ (v_c,\ v_{t2},\ (\sigma'_3,\ m)) \in \mathcal{V}\llparenthesis\varphi\tau_2\rrparenthesis_{L_2} \tag{D4}$$

We can apply corollary 22 to eq. (C1), eq. (D2), eq. (C3), eq. (D4) (note that $\mathbb{C} \trianglelefteq \tau_2$) and obtain v_t, $\vec{v^i} = [v^i_1, \ldots, v^i_n]$, $\vec{v^c} = [v^c_1, \ldots, v^c_n]$ and $\vec{l} = [l_1, \ldots, l_n]$, $\vec{l'} = [l'_1, \ldots, l'_n]$ both with all the elements pairwise distinct and furthermore for all $i \in [1, n]$, $l_i \in L_1$, $l_i \notin \texttt{dom}(\sigma_3)$, $l'_i \in L_2$ and $l'_i \notin \texttt{dom}(\sigma'_3)$, such that:

$$\underline{\mathbf{deep}}(!l\ v'_t,\ \tau_2),\ \sigma_2 \Downarrow^{L_1} (v, \vec{l}),\ \sigma_3[\langle\vec{l} \mapsto \vec{v^i}\rangle],\ \varnothing,\ c_3 + 2 + \texttt{cost}_{\texttt{deep}}(\tau) \tag{E1}$$

$$\underline{\mathbf{deep'}}(\beta_2(!l)\ \beta_2(v'_t),\ \tau_2),\ \sigma'_2 \Downarrow^{L_2} \vec{l'},\ \sigma'_3[\langle\vec{l'} \mapsto \vec{v^c}\rangle],\ \varnothing,\ c'_3 + 2 + \texttt{cost}_{\texttt{deepref}}(\tau) \tag{E2}$$

$$(v_i,\ v_c,\ v_t,\ (\sigma_3[\langle\vec{l} \mapsto \vec{v^i}\rangle],\ \sigma'_3[\beta_2 \otimes \langle\vec{l'} \mapsto \vec{v^c}\rangle],\ \vec{l} \mapsto \vec{l'},\ k)) \in \mathcal{V}[\![\tau_2]\!] \tag{E3}$$

We can now show the goals:

1. From eq. (A1), eq. (B1) and eq. (D1) we can derive that

$$\theta_c(e_1\ e_2) \Downarrow v_c,\ j'_1 + j'_2 + j'_3 + 1$$

2. From eq. (A2), eq. (B2) and eq. (E1) we can derive that

$$\begin{gathered}\theta_t\ulcorner e\urcorner,\ \sigma'_i \\ \Downarrow^{L_1} \\ v_t,\ \sigma_3,\ D_1 + D_2 + (\vec{l},\ \lambda().\, \underline{\mathbf{deep'}}(l\ v'_t,\ \tau_2)),\ c_1 + c_2 + c_3 + 9 + \texttt{cost}_{\texttt{deep}}(\tau_2)\end{gathered}$$

3. From eq. (A3) and eq. (B3) and using lemma 11 and lemma 13 we can derive that

$$D_1 + D_2,\ \sigma'_c,\ \sigma_2,\ \beta' \rightsquigarrow \sigma'_2,\ \beta_2,\ c'_1 + c'_2$$

There are two possible cases

- $\sigma'_2 = \wedge \beta_2 = \emptyset$

  In this case we can derive that

  $$(\vec{l},\ \lambda().\, \underline{\mathbf{deep}'}(!l\ v'_t,\ \tau_2)),\ \sigma'_2,\ \sigma_3,\ \beta_2 \rightsquigarrow \sigma'_2,\ \beta_2,\ 0$$

  Using lemma 11 and lemma 13 on the two change propagation judgments we can derive that

  $$D_1 + D_2 + (\vec{l},\ \lambda().\, \underline{\mathbf{deep}'}(!l\ v'_t,\ \tau_2)),\ \sigma'_c,\ \sigma_3,\ \beta' \rightsquigarrow \sigma'_2,\ \beta_2,\ c'_1 + c'_2$$

- $\sigma'_2 \neq \emptyset \vee \beta_2 \neq \emptyset$

  Using eq. (E2) we can derive that

  $$\beta_2(\lambda().\, \underline{\mathbf{deep}'}(!l\ v'_t,\ \tau_2))\ (),\ \sigma'_2 \Downarrow^{L_2} 0,\ \vec{l'},\ \sigma'_3[\langle \vec{l'} \mapsto \vec{v^c} \rangle],\ \varnothing 0 c'_3 + 3 + \texttt{cost}_{\texttt{deepref}}(\tau)$$

  In addition, we can easily see that $\beta_2 \otimes (\vec{l} \mapsto \vec{l'})$ is well defined since $\texttt{dom}(\beta_2) \subseteq \texttt{dom}(\sigma_3)$, $\texttt{dom}'(\beta_2) \subseteq \texttt{dom}(\sigma'_3)$ and for all $l \in \vec{l}$, $l \in L_1$, $l \notin \texttt{dom}(\sigma_3)$, $l \in \vec{l'}$, $l \in L_2$, $l \notin \texttt{dom}(\sigma'_3)$. By the definition of change propagation algorithm we can derive that

  $$(\vec{l},\ \lambda().\, \underline{\mathbf{deep}'}(!l\ v'_t,\ \tau_2)),\ \sigma'_2,\ \sigma_3,\ \beta_2 \rightsquigarrow \sigma'_3[\langle \vec{l'} \mapsto \vec{v^c} \rangle],\ \beta_2 \otimes (\vec{l} \mapsto \vec{l'}),\ c'_3 + 3 + \texttt{cost}_{\texttt{deepref}}(\tau_2)$$

  Using [**?**] and [**?**] on the two change propagation judgments above we can derive that

  $$D_1 + D_2 + (\vec{l},\ \lambda().\, \underline{\mathbf{deep}'}(l\ v'_t,\ \tau_2)),\ \sigma'_2,\ \sigma_3,\ \beta' \rightsquigarrow \sigma'_3[\langle \vec{l'} \mapsto \vec{v^c} \rangle],\ \beta_2 \otimes (\vec{l} \mapsto \vec{l'}),\ c'_1 + c'_2 + c'_3 + 3 + \texttt{cost}_{\texttt{deepref}}(\tau_2)$$

4. We should distinguish the following cases:

   - $\sigma'_2 = \emptyset \wedge \beta_2 = \emptyset$ From eq. (A4) and eq. (B4) we can derive that

     $$c'_1 + c'_2 \leq \varphi\kappa$$

   - $\sigma'_2 \neq \emptyset \vee \beta_2 \neq \emptyset$

     From eq. (A4), eq. (B4) and eq. (D3) we can derive that

     $$c'_1 + c'_2 + c'_3 + 3 + \texttt{cost}_{\texttt{deepref}}(\tau_2) \leq \varphi\kappa$$

5. We should distinguish the following cases:

   - $\sigma'_2 = \emptyset \wedge \beta_2 = \emptyset$

     Using eq. (A5) and eq. (B5) we derive that

     $$\texttt{dom}(\beta_2) \setminus \texttt{dom}(\beta') \subseteq \texttt{dom}(\sigma_3) \setminus \texttt{dom}(\sigma'_i) \wedge \texttt{dom}'(\beta_2) \setminus \texttt{dom}'(\beta') \subseteq \texttt{dom}(\sigma'_2) \setminus \texttt{dom}(\sigma'_c)$$

- $\sigma'_2 \neq \emptyset \vee \beta_2 \neq \emptyset$ Using eq. (A5) and eq. (B5) we derive that

$$\texttt{dom}(\beta_2 \otimes (\vec{l} \mapsto \vec{l'})) \setminus \texttt{dom}(\beta') \subseteq \texttt{dom}(\sigma_3[\langle \vec{l} \mapsto \vec{v^i} \rangle]) \setminus \texttt{dom}(\sigma'_i) \wedge \texttt{dom}'(\beta_2 \otimes (\vec{l} \mapsto \vec{l'})) \setminus \texttt{dom}(\beta') \subseteq \texttt{dom}(\sigma'_3[\langle \vec{l'} \mapsto \vec{v^c} \rangle$$

6. We should again distinguish the following cases:
   - $\sigma'_2 = \emptyset \wedge \beta_2 = \emptyset$
     Using the fact that evaluation is deterministic we can show that $v_{t2} = v_{t1}$. Then, using eq. (C3), eq. (D4) and lemma 17 we derive that

$$(v_i,\ v_c,\ v_t,\ (\sigma_3,\ \sigma'_2,\ \beta_2,\ m - j_i)) \in \mathcal{V}[\![\varphi \tau_2]\!]$$

   - $\sigma'_2 \neq \emptyset \vee \beta_2 \neq \emptyset$
     From eq. (E3) using lemma 15 we derive that

$$(v_i,\ v_c,\ v_t,\ (\sigma_3[\langle \vec{l} \mapsto \vec{v^i} \rangle],\ \sigma'_3[\langle \vec{l'} \mapsto \vec{v^c} \rangle],\ \beta_2 \otimes (\vec{l} \mapsto \vec{l'}),\ m - j_i)) \in \mathcal{V}[\![\varphi \tau_2]\!]$$

- $$\frac{\Delta; \Phi; \Gamma \vdash_{\epsilon}^{\kappa'} e : \tau \hookrightarrow \ulcorner e \urcorner \qquad \forall x \in \Gamma \quad \Delta; \Phi \vdash \Gamma(x) \sqsubseteq \Gamma(x)^{\downarrow\square} \qquad \Delta; \Phi \models \kappa \doteq (\epsilon = \mathbb{S}\ \texttt{?}\ 0\ :\ \kappa')}{\Delta; \Phi; \Gamma, \Gamma' \vdash_{\epsilon}^{\kappa} e : \tau^{\downarrow\square} \hookrightarrow \texttt{drop}(\underline{\textbf{conv}}(\ulcorner e \urcorner,\ \tau,\ \tau^{\downarrow\square}))}\ \textbf{nochange}$$

We pick arbitrary $j_i < m$, v_i, $\sigma'_i \sqsupseteq^{L_1} \sigma_i$, $\sigma'_c \sqsupseteq^{L_2} \sigma_c$, $\beta' \geq \beta$ such that

$$\theta_i(e_1\ e_2) \Downarrow v_i,\ j_i \tag{1}$$

$$\texttt{dom}(\beta') \setminus \texttt{dom}(\beta) \subseteq \texttt{dom}(\sigma'_i) \setminus \texttt{dom}(\sigma_i) \wedge \texttt{dom}'(\beta') \setminus \texttt{dom}'(\beta) \subseteq \texttt{dom}(\sigma'_c) \setminus \texttt{dom}(\sigma_c) \tag{2}$$

Using lemma 15 we can show that

$$(\theta_i,\ \theta_c,\ \theta_t,\ (\sigma'_i,\ \sigma'_c,\ \beta',\ m)) \in \mathcal{G}[\![\varphi\Gamma; \varphi\Gamma']\!]$$

By the definition of $\mathcal{G}[\![\cdot]\!]$ we can derive that

$$(\theta_i,\ \theta_c,\ \theta_t,\ (\sigma'_i,\ \sigma'_c,\ \beta',\ m)) \in \mathcal{G}[\![\varphi\Gamma]\!]$$

Using lemma 26 we obtain

$$(\theta_i,\ \theta_c,\ \theta_t,\ (\sigma'_i,\ \emptyset,\ \emptyset,\ m)) \in \mathcal{G}[\![\varphi\Gamma]\!]$$

We instantiate the inductive hypothesis with the above and we derive

$$(\theta_i e,\ \theta_c e,\ \theta_t \ulcorner e \urcorner,\ (\sigma'_i,\ \emptyset,\ \emptyset,\ m)) \in \mathcal{E}[\![\varphi\tau]\!]^{\varphi\kappa'}$$

We instantiate the above statement with j_i, v_i, σ'_i, $\emptyset$, $\emptyset$ and eq. (1) and we obtain v_c, j_c, v_t, σ_f, D, c, σ'_f, β_f, c' such that:

$$\theta_c e \Downarrow v_c,\ j_c \tag{A1}$$

$$\theta_t\ulcorner e\urcorner,\ \sigma'_i \Downarrow^{L_1} v_t,\ \sigma_f,\ D,\ c \tag{A2}$$

$$D,\ \emptyset,\ \sigma_f,\ \emptyset \leadsto \sigma'_f,\ \beta_f,\ c' \tag{A3}$$

$$\models c' \dot{\leq} \kappa \tag{A4}$$

$$\texttt{dom}(\beta_f) \setminus \texttt{dom}(\emptyset) \subseteq \texttt{dom}(\sigma_f) \setminus \texttt{dom}(\sigma'_i) \wedge \texttt{dom}'(\beta_f) \setminus \texttt{dom}(\emptyset) \subseteq \texttt{dom}(\sigma'_f) \setminus \texttt{dom}(\sigma'_c) \tag{A5}$$

and

$$(v_i,\ v_c,\ v'_t,\ (\sigma_f,\ \sigma'_f,\ \beta_f,\ m - j_i)) \in \mathcal{V}[\![\tau]\!] \tag{A6}$$

From eq. (A3) and the definition of change propagation we can derive that $\sigma'_f = \emptyset$, $\beta_f = \emptyset$ and $c' = 0$. We instantiate lemma 23 with eq. (A2) and eq. (A6) and obtain v'_t such that

$$\underline{\mathbf{conv}}(\theta_t\ulcorner e\urcorner,\ \tau,\ \tau^{\downarrow\square}),\ \sigma'_i \Downarrow^{L_1} v'_t,\ \sigma_f,\ D,\ c + \texttt{cost}_{\texttt{conv}}(\tau, \tau^{\downarrow\square}) \tag{B1}$$

and

$$\forall\beta,\ (v_i,\ v_c,\ v'_t,\ (\sigma_f,\ \sigma'_c,\ \beta,\ m - j_i)) \in \mathcal{V}[\![\tau^{\downarrow\square}]\!] \tag{B2}$$

We can now show the goals:

1. It follows immediately from eq. (A1).
2. Using eq. (B1) we can show

$$\texttt{drop}(\underline{\mathbf{conv}}(\ulcorner e\urcorner,\ \tau,\ \tau^{\downarrow\square})),\ \sigma'_i \Downarrow^{L_1} v'_t,\ \sigma_f,\ \varnothing,\ c + \texttt{cost}_{\texttt{conv}}(\tau, \tau^{\downarrow\square}) + 1$$

3. We can easily show, using the definition of change propagation, that

$$\varnothing,\ \sigma'_c,\ \sigma_f,\ \beta' \leadsto \sigma'_c,\ \beta',\ 0$$

4. It is trivially true since $\texttt{dom}(\beta') \setminus \texttt{dom}(\beta') = \emptyset$ and $\texttt{dom}'(\beta') \setminus \texttt{dom}'(\beta') = \emptyset$
5. We can instantiate eq. (B2) with β' and obtain

$$(v_i,\ v_c,\ v'_t,\ (\sigma_f,\ \sigma'_c,\ \beta',\ m - j_i)) \in \mathcal{V}[\![\tau^{\downarrow\square}]\!]$$

- $$\frac{\Delta;\Phi;\Gamma, f : (\tau_1 \xrightarrow{\delta(\kappa)} \tau_2)^\square, x : \tau_1 \vdash^\kappa_\delta e : \tau_2 \hookrightarrow \ulcorner e\urcorner \qquad \forall x \in \Gamma \quad \Delta;\Phi \models \Gamma(x) \sqsubseteq \Gamma(x)^{\downarrow\square}}{\Delta;\Phi;\Gamma,\Gamma' \vdash^0_\epsilon \texttt{fix}\ f(x).\,e : (\tau_1 \xrightarrow{\delta(\kappa)} \tau_2)^\square \hookrightarrow \texttt{fix}\ f(x).\,\ulcorner e\urcorner}\ \textbf{fix2}$$

We pick arbitrary $j_i < m$, v_i, $\sigma'_i \sqsupseteq^{L_1} \sigma_i$, $\sigma'_c \sqsupseteq^{L_2} \sigma_c$, $\beta' \geq \beta$ such that

$$\theta_i(\texttt{fix}\ f(x).\,e) \Downarrow v_i,\ j_i$$

$$\texttt{dom}(\beta') \setminus \texttt{dom}(\beta) \subseteq \texttt{dom}(\sigma'_i) \setminus \texttt{dom}(\sigma_i)$$

and

$$\texttt{dom}'(\beta') \setminus \texttt{dom}'(\beta) \subseteq \texttt{dom}(\sigma_c') \setminus \texttt{dom}(\sigma_c)$$

By inversion of the evaluation relation we derive that $j_i = 0$ and $v_i = \theta_i(\texttt{fix}\ f(x).\, e)$

The goals are proved as follows:

1. We can easily derive that
$$\theta_c(\texttt{fix}\ f(x).\, e) \Downarrow \theta_c(\texttt{fix}\ f(x).\, e),\ 0$$
2. Similarly, we can show
$$\theta_t(\texttt{fix}\ f(x).\, e),\ \sigma_i' \Downarrow^{L_1} \theta_t(\texttt{fix}\ f(x).\ulcorner e \urcorner),\ \sigma_i',\ \varnothing,\ 0$$
3. By the definition of the change propagation algorithm we can easily obtain that:
$$\varnothing,\ \sigma_c',\ \sigma_i',\ \beta' \rightsquigarrow \sigma_c',\ \beta',\ 0$$
4. It is obvious that $0 \leq 0$
5. It is trivially true since $\texttt{dom}(\beta') \setminus \texttt{dom}(\beta') = \emptyset$ and $\texttt{dom}'(\beta') \setminus \texttt{dom}'(\beta') = \emptyset$
6. Finally, we need to show
$$(\theta_i(\texttt{fix}\ f(x).\, e),\ \theta_c(\texttt{fix}\ f(x).\, e),\ \theta_t(\texttt{fix}\ f(x).\ulcorner e \urcorner),\ (\sigma_i',\ \sigma_c',\ \beta',\ m)) \in \mathcal{V}[\![(\varphi\tau_1 \xrightarrow{\varphi(\delta(\kappa))} \varphi\tau_2)^{\square}]\!]$$
Or, equivalently
$$(\theta_i(\texttt{fix}\ f(x).\, e),\ \theta_c(\texttt{fix}\ f(x).\, e),\ \theta_t(\texttt{fix}\ f(x).\ulcorner e \urcorner),\ (\sigma_i',\ \emptyset,\ \emptyset,\ m)) \in \mathcal{V}_A[\![\varphi\tau_1 \xrightarrow{\varphi(\delta(\kappa))} \varphi\tau_2]\!]$$
From the hypotheses we know that
$$(\theta_i,\ \theta_c,\ \theta_t,\ (\sigma_i,\ \sigma_c,\ \beta',\ m)) \in \mathcal{G}[\![\varphi\Gamma]\!]$$
Using lemma 26 we obtain
$$(\theta_i,\ \theta_c,\ \theta_t,\ (\sigma_i,\ \sigma_c,\ \emptyset,\ m)) \in \mathcal{G}[\![\varphi\Gamma]\!]$$
We can now show the goal by case analysis on δ and induction on m. The proof is similar to the one of **fix1** rule.

- $$\frac{\Delta; \Phi; \Gamma \vdash_{\epsilon}^{\kappa_1} e_1 : \tau^{\downarrow\square} \hookrightarrow \ulcorner e_1 \urcorner \qquad \Delta; \Phi; \Gamma \vdash_{\epsilon}^{\kappa_2} e_2 : \texttt{list}\,[n]^{\alpha}\ \tau \hookrightarrow \ulcorner e_2 \urcorner \qquad \Delta; \Phi \models \kappa \doteq \kappa_1 \mathbin{\hat{+}} \kappa_2}{\Delta; \Phi; \Gamma \vdash_{\epsilon}^{\kappa} e_1 :: e_2 : \texttt{list}\left[n \mathbin{\hat{+}} 1\right]^{\alpha}\ \tau \hookrightarrow \texttt{inl}\ \ulcorner e_1 \urcorner :: \ulcorner e_2 \urcorner}\ \textbf{cons1}$$

By the induction hypothesis applied on the premises we get:

$$(\theta_i e_1,\ \theta_c e_1,\ \theta_t \ulcorner e_1 \urcorner,\ (\sigma_i,\ \sigma_c,\ \beta,\ m)) \in \mathcal{E}[\![\varphi\tau^{\downarrow\square}]\!]^{\varphi\kappa_1} \tag{A}$$

$$(\theta_i e_2,\ \theta_c e_2,\ \theta_t \ulcorner e_2 \urcorner,\ (\sigma_i,\ \sigma_c,\ \beta,\ m)) \in \mathcal{E}[\![\texttt{list}\,[\varphi n]^{\varphi\alpha}\ \varphi\tau]\!]^{\varphi\kappa_2} \tag{B}$$

We pick arbitrary $j_i < m$, v, $\sigma_i' \sqsupseteq^{L_1} \sigma_i$, $\sigma_c' \sqsupseteq^{L_2} \sigma_c$, $\beta' \geq \beta$ such that

$$\theta_i(e_1 :: e_2) \Downarrow v,\ j_i$$

$$\texttt{dom}(\beta') \setminus \texttt{dom}(\beta) \subseteq \texttt{dom}(\sigma_i') \setminus \texttt{dom}(\sigma_i)$$

and

$$\texttt{dom}'(\beta') \setminus \texttt{dom}'(\beta) \subseteq \texttt{dom}(\sigma_c') \setminus \texttt{dom}(\sigma_c)$$

By inversion of the evaluation relation we derive the following:

$$\theta_i e_1 \Downarrow v_i,\ j_1 \tag{1}$$

$$\theta_i e_2 \Downarrow vs_i,\ j_2 \tag{2}$$

$$e_i'[x/v_i'][f/\texttt{fix}\ f(x).\,e_i'] \Downarrow v_i,\ j_3 \tag{3}$$

$$j_i = j_1 + j_2$$

$$v = v_i :: vs_i$$

for some v_i, vs_i, j_1 and j_2.

We instantiate eq. (A) with j_1 (note that $j_1 < m$), σ_i', σ_c', β' (note that $\sigma_i' \sqsupseteq^{L_1} \sigma_i$, $\sigma_c' \sqsupseteq^{L_2} \sigma_c$, $\beta' \geq \beta$ and the preconditions for the bijection hold), and eq. (1) and we obtain v_c, j_1', v_t, σ_1, D_1, c_1, σ_1', β_1, c_1' such that:

$$\theta_c e_1 \Downarrow v_c,\ j_1' \tag{A1}$$

$$\theta_t \ulcorner e_1 \urcorner,\ \sigma_i \Downarrow^{L_1} v_t,\ \sigma_1,\ D_1,\ c_1 \tag{A2}$$

$$D_1,\ \sigma_c',\ \sigma_1,\ \beta' \rightsquigarrow \sigma_1',\ \beta_1,\ c_1' \tag{A3}$$

$$c_1' \leq \varphi\kappa_1 \tag{A4}$$

$$\texttt{dom}(\beta_1) \setminus \texttt{dom}(\beta') \subseteq \texttt{dom}(\sigma_1) \setminus \texttt{dom}(\sigma_i') \wedge \texttt{dom}'(\beta_1) \setminus \texttt{dom}'(\beta') \subseteq \texttt{dom}(\sigma_1') \setminus \texttt{dom}(\sigma_c') \tag{A5}$$

and

$$(v_i,\ v_c,\ v_t,\ (\sigma_1,\ \sigma_1',\ \beta_1,\ m - j_1)) \in \mathcal{V}[\![\varphi\tau^{\downarrow\Box}]\!] \tag{A6}$$

We now instantiate eq. (B) with j_2 (note that $j_2 < m$) σ_1, σ_1', β_1 (note that $\sigma_1 \sqsupseteq^{L_1} \sigma_i'$, $\sigma_1' \sqsupseteq^{L_2} \sigma_c'$, $\beta_1 \geq \beta'$) eq. (A6) and eq. (2), and we obtain vs_c, j_2', vs_t, σ_2, D_2, c_2, σ_2', β_2, c_2' such that:

$$\theta_c e_2 \Downarrow vs_c,\ j_2' \tag{B1}$$

$$\theta_t \ulcorner e_2 \urcorner,\ \sigma_1 \Downarrow^{L_1} vs_t,\ \sigma_2,\ D_2,\ c_2 \tag{B2}$$

$$D_2,\ \sigma_1',\ \sigma_2,\ \beta_1 \rightsquigarrow \sigma_2',\ \beta_2,\ c_2' \tag{B3}$$

$$c_2' \leq \varphi\kappa_2 \tag{B4}$$

$$\mathtt{dom}(\beta_2) \setminus \mathtt{dom}(\beta_1) \subseteq \mathtt{dom}(\sigma_2) \setminus \mathtt{dom}(\sigma_1) \wedge \mathtt{dom}'(\beta_2) \setminus \mathtt{dom}'(\beta_1) \subseteq \mathtt{dom}(\sigma_2') \setminus \mathtt{dom}(\sigma_1') \tag{B5}$$

and

$$(vs_i,\ vs_c,\ vs_t,\ (\sigma_2,\ \sigma_2',\ \beta_2,\ m - j_2)) \in \mathcal{V}[\![\mathtt{list}\,[\varphi n]^{\varphi\alpha}\ \varphi\tau]\!] \tag{B6}$$

We can now show the goals:

1. From eq. (A1) and eq. (B1) we can derive that

$$\theta_c(e_1 :: e_2) \Downarrow v_c :: vs_c,\ j_1' + j_2'$$

2. From eq. (A2) and eq. (B2) we can derive that

$$\theta_t(\mathtt{inl}\ \ulcorner e_1 \urcorner :: \ulcorner e_2 \urcorner),\ \sigma_i' \Downarrow^{L_1} \mathtt{inl}\ v_t :: vs_t,\ \sigma_2,\ D_1 + D_2,\ c_1 + c_2$$

3. From eq. (A3) and eq. (B3) using lemma 13 and lemma 11 we can derive that

$$D_1 + D_2,\ \sigma_i',\ \sigma_2 \rightsquigarrow \beta',\ \sigma_2',\ \beta_2 c_1' + c_2'$$

4. From eq. (A4), eq. (B4) we can derive that

$$c_1' + c_2' \leq \varphi(\kappa_1 + \kappa_2)$$

5. From eq. (A5) and eq. (B5) we can derive that

$$\mathtt{dom}(\beta_2) \setminus \mathtt{dom}(\beta') \subseteq \mathtt{dom}(\sigma_2) \setminus \mathtt{dom}(\sigma_i') \wedge \mathtt{dom}'(\beta_2) \setminus \mathtt{dom}'(\beta') \subseteq \mathtt{dom}(\sigma_2') \setminus \mathtt{dom}(\sigma_c')$$

6. We have to show that

$$(v_i :: vs_i,\ v_c :: vs_c,\ \mathtt{inl}\ v_t :: vs_t,\ (\sigma_2,\ \sigma_2',\ \beta_2,\ m - j_i)) \in \mathcal{V}[\![\mathtt{list}\,[\varphi n + 1]^{\varphi\alpha}\ \varphi\tau]\!]$$

   It suffices to show that

$$(v_i,\ v_c,\ v_t,\ (\sigma_2,\ \sigma_2',\ \beta_2,\ m - j_i)) \in \mathcal{V}[\![\varphi\tau^{\downarrow\square}]\!]$$

   which we get by applying lemma 15 to eq. (A6), and

$$(vs_i,\ vs_c,\ vs_t,\ (\sigma_2,\ \sigma_2',\ \beta_2,\ m - j_i)) \in \mathcal{V}[\![\mathtt{list}\,[\varphi n]^{\varphi\alpha}\ \varphi\tau]\!]$$

   which we get by applying lemma 15 to eq. (B6)

- $$\frac{\begin{array}{c}\Delta;\Phi;\Gamma \vdash_{\epsilon}^{\kappa_e} e : \mathtt{list}\,[n]^{\alpha}\ \tau \hookrightarrow \ulcorner e \urcorner \qquad \Delta;\Phi \wedge n\ \doteq\ 0;\Gamma \vdash_{\epsilon}^{\kappa'} e_1 : \tau' \hookrightarrow \ulcorner e_1 \urcorner \\ \Delta, i :: \iota;\Phi \wedge n \doteq i+1;\Gamma, h : \tau^{\downarrow\square}, tl : \mathtt{list}\,[i]^{\alpha}\ \tau \vdash_{\epsilon}^{\kappa'} e_2 : \tau' \hookrightarrow \ulcorner e_2 \urcorner_l \\ \Delta, i :: \iota, \beta :: \iota;\Phi \wedge n \doteq i+1 \wedge \alpha \doteq \beta+1;\Gamma, h : \tau, tl : \mathtt{list}\,[i]^{\beta}\ \tau \vdash_{\epsilon}^{\kappa'} e_2 : \tau' \hookrightarrow \ulcorner e_2 \urcorner_r \\ \Delta;\Phi \models \kappa \doteq \kappa_e\ \hat{+}\ \kappa'\ \hat{+}\ (\epsilon = \mathbb{C})\ \mathtt{?}\ 1\ :\ 0\end{array}}{\Delta;\Phi;\Gamma \vdash_{\epsilon}^{\kappa} \mathtt{case}_{\mathtt{L}}\ e\ \mathtt{of}\ \mathtt{nil}\ \rightarrow\ e_1 \mid \mathtt{cons}(h, tl)\ \rightarrow\ e_2 : \tau' \hookrightarrow \begin{array}{l}\mathtt{case}_{\mathtt{L}}\ \ulcorner e \urcorner\ \mathtt{of} \\ \mid\ \mathtt{nil}\ \rightarrow\ \ulcorner e_1 \urcorner \\ \mid\ \mathtt{cons}(s, tl)\ \rightarrow\ \mathtt{case}(s, h.\ulcorner e_2 \urcorner_l, h.\ulcorner e_2 \urcorner_r)\end{array}}\ \textbf{caseL}$$

By the induction hypothesis applied on the first premise :

$$(\theta_i e,\ \theta_c e,\ \theta_t \ulcorner e \urcorner,\ (\sigma_i,\ \sigma_c,\ \beta,\ m)) \in \mathcal{E}[\![\mathtt{list}\,[\varphi n]^{\varphi\alpha}\ \varphi\tau]\!]^{\varphi\kappa_e} \tag{A}$$

Let $\mathtt{dom}(\beta) \subseteq \mathtt{dom}(\sigma_i)$ and $\mathtt{dom}'(\beta) \subseteq \mathtt{dom}(\sigma_c)$. We pick arbitrary $j_i < m$, v, $\sigma_i' \sqsupseteq^{L_1} \sigma_i$, $\sigma_c' \sqsupseteq^{L_2} \sigma_c$, $\beta' \geq \beta$ such that

$$\theta_i(\mathtt{case}_{\mathtt{L}}\ e_e\ \mathtt{of}\ \mathtt{nil}\ \rightarrow\ e_1 \mid \mathtt{cons}(h, tl)\ \rightarrow\ e_2) \Downarrow v_i,\ j_i$$

$$\mathtt{dom}(\beta') \setminus \mathtt{dom}(\beta) \subseteq \mathtt{dom}(\sigma_i') \setminus \mathtt{dom}(\sigma_i)$$

and

$$\mathtt{dom}'(\beta') \setminus \mathtt{dom}'(\beta) \subseteq \mathtt{dom}(\sigma_c') \setminus \mathtt{dom}(\sigma_c)$$

We invert the evaluation relation and we have the following cases:

► **case-nil**

$$\theta_i e \Downarrow [],\ j_1 \tag{1}$$

$$\theta_i e_1 \Downarrow v_i,\ j_2 \tag{2}$$

$$j_i = j_1 + j_2 + 1$$

We instantiate eq. (A) with j_1 (note that $j_1 < m$), σ_i', σ_c', β' (note that $\sigma_i' \sqsupseteq^{L_1} \sigma_i$, $\sigma_c' \sqsupseteq^{L_2} \sigma_c$, $\beta' \geq \beta$, and the preconditions for the bijection hold), and eq. (1) and we obtain v_c', j_1', v_t', σ_1, D_1, c_1, σ_1', β_1, c_1' such that:

$$\theta_c e_1 \Downarrow v_c',\ j_1' \tag{A1}$$

$$\theta_t \ulcorner e_1 \urcorner,\ \sigma_i \Downarrow^{L_1} v_t',\ \sigma_1,\ D_1,\ c_1 \tag{A2}$$

$$D_1,\ \sigma_c',\ \sigma_1,\ \beta' \rightsquigarrow \sigma_1',\ \beta_1,\ c_1' \tag{A3}$$

$$c_1' \leq \varphi\kappa_e \tag{A4}$$

$$\mathtt{dom}(\beta_1) \setminus \mathtt{dom}(\beta') \subseteq \mathtt{dom}(\sigma_1) \setminus \mathtt{dom}(\sigma_i') \wedge \mathtt{dom}'(\beta_1) \setminus \mathtt{dom}'(\beta') \subseteq \mathtt{dom}(\sigma_1') \setminus \mathtt{dom}(\sigma_c') \tag{A5}$$

and

$$([],\ v'_c,\ v'_t,\ (\sigma_1,\ \sigma'_1,\ \beta_1,\ m - j_1)) \in \mathcal{V}[\![\texttt{list}\,[\varphi n]^{\varphi\alpha}\ \varphi\tau]\!]$$

From the last statement we derive that $v'_c = []$, $v'_t = []$, $\varphi n = 0$ and $\varphi\alpha = 0$. Consequently, we can derive that $\models \varphi(\Phi \wedge n \doteq 0)$. We instantiate the inductive hypothesis for the second premise with the above and we an derive

$$(\theta_i e_1,\ \theta_c e_1,\ \theta_t \ulcorner e_1 \urcorner,\ (\sigma_i,\ \sigma_c,\ \beta,\ m)) \in \mathcal{E}[\![\varphi\tau']\!]^{\varphi\kappa_1} \tag{B}$$

We now instantiate eq. (B) with j_2 (note that $j_2 < m$) σ_1, σ'_1, β_1 (note that $\sigma_1 \sqsupseteq^{L_1} \sigma'_i$, $\sigma'_1 \sqsupseteq^{L_2} \sigma'_c$), eq. (A5), and eq. (2) and we obtain v_c, j'_2, v_t, σ_2, D_2, c_2, σ'_2, β_2, c'_2 such that:

$$\theta_c e_2 \Downarrow v_c,\ j'_2 \tag{B1}$$

$$\theta_t \ulcorner e_2 \urcorner,\ \sigma_1 \Downarrow^{L_1} v_t,\ \sigma_2,\ D_2,\ c_2 \tag{B2}$$

$$D_2,\ \sigma'_1,\ \sigma_2,\ \beta_1 \rightsquigarrow \sigma'_2,\ \beta_2,\ c'_2 \tag{B3}$$

$$c'_2 \leq \varphi\kappa' \tag{B4}$$

$$\texttt{dom}(\beta_2) \setminus \texttt{dom}(\beta_1) \subseteq \texttt{dom}(\sigma_2) \setminus \texttt{dom}(\sigma_1) \wedge \texttt{dom}'(\beta_2) \setminus \texttt{dom}'(\beta_1) \subseteq \texttt{dom}(\sigma'_2) \setminus \texttt{dom}(\sigma'_1) \tag{B5}$$

and

$$(v_i,\ v_c,\ v_t,\ (\sigma_2,\ \sigma'_2,\ \beta_2,\ m - j_2)) \in \mathcal{V}[\![\varphi\tau']\!] \tag{B6}$$

We can now show the goals:

1. From eq. (A1) and eq. (B1) we can derive that

$$\theta_c(\texttt{case}_\texttt{L}\ e\ \texttt{of nil}\ \rightarrow\ e_1 \mid \texttt{cons}(h, tl)\ \rightarrow\ e_2) \Downarrow v_c,\ j'_1 + j'_2 + 1$$

2. From eq. (A2) and eq. (B2) we can derive that

$$\begin{gathered}\theta_t(\texttt{case}_\texttt{L}\ \ulcorner e \urcorner\ \texttt{of nil}\ \rightarrow\ \ulcorner e_1 \urcorner \mid \texttt{cons}(s, tl)\ \rightarrow\ \ldots),\ \sigma'_i \\ \Downarrow^{L_1} \\ v_t,\ \sigma_2,\ D_1 + D_2,\ c_1 + c_2 + 1\end{gathered}$$

3. From eq. (A3) and eq. (B3) using lemma 13 and lemma 11 we can derive that

$$D_1 + D_2,\ \sigma'_c,\ \sigma_2,\ \beta' \rightsquigarrow \sigma'_2,\ \beta_2,\ c'_1 + c'_2$$

4. From eq. (A5) and eq. (B5) we can derive that

$$\texttt{dom}(\beta_2) \setminus \texttt{dom}(\beta') \subseteq \texttt{dom}(\sigma_2) \setminus \texttt{dom}(\sigma'_i) \wedge \texttt{dom}'(\beta_2) \setminus \texttt{dom}'(\beta') \subseteq \texttt{dom}(\sigma'_2) \setminus \texttt{dom}(\sigma'_c)$$

5. From eq. (B6) and lemma 15 we derive that

$$(v_i,\ v_c,\ v_t,\ (\sigma_2,\ \sigma_2',\ \beta_2,\ m - j_i)) \in \mathcal{V}[\![\varphi\tau']\!]$$

► **case-cons**

$$\theta_i e \Downarrow v_i' :: vs_i,\ j_1 \tag{1}$$

$$\theta_i[h \mapsto v_i', tl \mapsto vs_i]e_2 \Downarrow v_i,\ j_2 \tag{2}$$

$$j_i = j_1 + j_2 + 1$$

We instantiate eq. (A) with j_1 (note that $j_1 < m$), σ_i', σ_c', β' (note that $\sigma_i' \sqsupseteq^{L_1} \sigma_i$, $\sigma_c' \sqsupseteq^{L_2} \sigma_c$, $\beta' \geq \beta$, and the preconditions for the bijection hold), and eq. (1) and we obtain v_c'', j_1', v_t'', σ_1, D_1, c_1, σ_1', β_1, c_1' such that

$$\theta_c e_1 \Downarrow v_c'',\ j_1' \tag{A1}$$

$$\theta_t \ulcorner e_1 \urcorner,\ \sigma_i \Downarrow^{L_1} v_t'',\ \sigma_1,\ D_1,\ c_1 \tag{A2}$$

$$D_1,\ \sigma_c',\ \sigma_1,\ \beta' \rightsquigarrow \sigma_1',\ \beta_1,\ c_1' \tag{A3}$$

$$c_1' \leq \varphi\kappa_e \tag{A4}$$

$$\texttt{dom}(\beta_1) \setminus \texttt{dom}(\beta') \subseteq \texttt{dom}(\sigma_1) \setminus \texttt{dom}(\sigma_i') \wedge \texttt{dom}'(\beta_1) \setminus \texttt{dom}'(\beta') \subseteq \texttt{dom}(\sigma_1') \setminus \texttt{dom}(\sigma_c') \tag{A5}$$

and

$$(v_i :: vs_i,\ v_c'',\ v_t'',\ (\sigma_1,\ \sigma_1',\ \beta_1,\ m - j_1)) \in \mathcal{V}[\![\texttt{list}\,[\varphi n]^{\varphi\alpha}\ \varphi\tau]\!] \tag{A6}$$

From the last statement we derive that $\varphi n = n' + 1$ for some integer n' and $v_c' = v_c' :: vs_c$ for some v_c', vs_c. We have the following two cases:

⋆ $v_t'' = \texttt{inl}\ v_t' :: vs_t$

From eq. (A6) we derive

$$(v_i',\ v_c',\ v_t',\ (\sigma_1,\ \sigma_1',\ \beta_1,\ m - j_1)) \in \mathcal{V}[\![\varphi\tau^{\downarrow\Box}]\!] \tag{A7.1}$$

$$(vs_i,\ vs_c,\ vs_t,\ (\sigma_1,\ \sigma_1',\ \beta_1,\ m - j_1)) \in \mathcal{V}[\![\texttt{list}\,[n']^{\varphi\alpha}\ \varphi\tau]\!] \tag{A7.2}$$

We can also show that $\models \varphi[i \mapsto n'](\Phi \wedge n \doteq i + 1)$. Using eq. (A7.1) and eq. (A7.2) we can show that

$$\begin{gathered}(\theta_i[h \mapsto v_i', tl \mapsto vs_i],\ \theta_c[h \mapsto v_c', tl \mapsto vs_c],\ \theta_t[h \mapsto v_t', tl \mapsto vs_t],\ (\sigma_1,\ \sigma_1',\ \beta_1,\ m - j_1)) \\ \in \mathcal{G}[\![\varphi[i \mapsto n']\Gamma, h : \varphi[i \mapsto n']\tau^{\downarrow\Box}, tl : \varphi[i \mapsto n'](\texttt{list}\,[i]^{\alpha}\ \tau)]\!]\end{gathered}$$

We instantiate the inductive hypothesis with the above statements and using lemma 15 we derive the following

$$(\theta_i e_2,\ \theta_c e_2,\ \theta_t \ulcorner e_2 \urcorner_l,\ (\sigma_1,\ \sigma_1',\ \beta_1,\ m - j_1 - 1)) \in \mathcal{E}[\![\texttt{list}\,[n']^{\varphi\alpha}\ \varphi\tau]\!]^{\varphi\kappa'} \tag{B}$$

(Note that the premises force that i does not appear free in κ') We now instantiate eq. (B) with j_2 (note that $j_2 < m - j_1 - 1$) σ_1, σ_1', β_1 (note that $\sigma_1 \sqsupseteq^{L_1} \sigma_i'$, $\sigma_1' \sqsupseteq^{L_2} \sigma_c'$, eq. (2), and eq. (A6) and we obtain v_c, j_2', v_t, σ_2, D_2, c_2, σ_2', β_2, c_2' such that:

$$\theta_c[h \mapsto v_c', tl \mapsto vs_c]e_2 \Downarrow v_c,\ j_2' \tag{B1}$$

$$\theta_t[h \mapsto v_t', tl \mapsto vs_t]\ulcorner e_2 \urcorner,\ \sigma_1 \Downarrow^{L_1} v_t,\ \sigma_2,\ D_2,\ c_2 \tag{B2}$$

$$D_2,\ \sigma_1',\ \sigma_2,\ \beta_1 \rightsquigarrow \sigma_2',\ \beta_2,\ c_2' \tag{B3}$$

$$c_2' \leq \varphi\kappa' \tag{B4}$$

$$\mathtt{dom}(\beta_2) \setminus \mathtt{dom}(\beta_1) \subseteq \mathtt{dom}(\sigma_2) \setminus \mathtt{dom}(\sigma_1) \wedge \mathtt{dom}'(\beta_2) \setminus \mathtt{dom}'(\beta_1) \subseteq \mathtt{dom}(\sigma_2') \setminus \mathtt{dom}(\sigma_1') \tag{B5}$$

and

$$(v_i,\ v_c,\ v_t,\ (\sigma_2,\ \sigma_2',\ \beta_2,\ m - j_1 - j_2 - 1)) \in \mathcal{V}[\![\varphi\tau']\!] \tag{B6}$$

We can now show the goals:

1. From eq. (A1) and eq. (B1) we can derive that

$$\theta_c(\mathtt{case_L}\ e\ \mathtt{of\ nil}\ \rightarrow\ e_1 \mid \mathtt{cons}(h, tl)\ \rightarrow\ e_2) \Downarrow v_c,\ j_1' + j_2' + 1$$

2. From eq. (A2) and eq. (B2) we can derive that

$$\begin{gathered}\theta_t(\mathtt{case_L}\ \ulcorner e \urcorner\ \mathtt{of\ nil}\ \rightarrow\ \ulcorner e_1 \urcorner \mid \mathtt{cons}(s, tl)\ \rightarrow\ \ldots),\ \sigma_i' \\ \Downarrow^{L_1} \\ v_t,\ \sigma_2,\ D_1 + D_2,\ c_1 + c_2 + 2\end{gathered}$$

3. From eq. (A3) and eq. (B3) using lemma 11 and lemma 13 we can derive that

$$D_1 + D_2,\ \sigma_i',\ \sigma_2,\ \beta' \rightsquigarrow \sigma_2',\ \beta_2,\ c_1' + c_2'$$

4. From eq. (A4) and eq. (B4) we can derive that

$$c_1' + c_2' \leq \varphi(\kappa_e + \kappa')$$

5. From eq. (A5) and eq. (B5) we can derive that

$$\mathtt{dom}(\beta_2) \setminus \mathtt{dom}(\beta') \subseteq \mathtt{dom}(\sigma_2) \setminus \mathtt{dom}(\sigma_i') \wedge \mathtt{dom}'(\beta_2) \setminus \mathtt{dom}'(\beta') \subseteq \mathtt{dom}(\sigma_2') \setminus \mathtt{dom}(\sigma_c')$$

6. Finally, using eq. (B6) we derive that

$$(v_i,\ v_c,\ v_t,\ (\sigma_2,\ \sigma_2',\ \beta_2,\ m - j_i)) \in \mathcal{V}[\![\varphi\tau']\!]$$

- $v'_t = \texttt{inr}\ v_t :: vs_t$

  From eq. (A6) we derive

$$(v'_i,\ v'_c,\ v'_t,\ (\sigma_1,\ \sigma'_1,\ \beta_1,\ m - j_1)) \in \mathcal{V}[\![\varphi\tau]\!] \tag{A7.1}$$

$$(vs_i,\ vs_c,\ vs_t,\ (\sigma_1,\ \sigma'_1,\ \beta_1,\ m - j_1)) \in \mathcal{V}[\![\texttt{list}\,[n']^{\varphi(\alpha-1)}\ \varphi\tau]\!] \tag{A7.2}$$

  and

$$\phi\alpha > 0$$

  or equivalently

$$\phi\alpha = m' + 1$$

  for some integer m'. We can also show that

$$\models \varphi[i \mapsto n', \beta \mapsto m'](\Phi \wedge n \doteq i + 1 \wedge \alpha \doteq \beta + 1)$$

  The rest of the proof is now similar to the one of the previous case.

□