Relational Reasoning
(Relationel ræsonnement)

Mathias Pedersen, 201808137

Bachelor Report (15 ECTS) in Computer Science
Advisor: Amin Timany

Department of Computer Science, Aarhus University

January 2022

AARHUS
UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

Abstract

This report investigates a notion of program equivalence called *contextual equivalence*, which has shown itself useful for example when justifying refactoring of code. Using the language *System F* as basis for discussion, we give two definitions of contextual equivalence, and show that they are equivalent.
Furthermore, we give a brief introduction to the theory of *logical relations* and present a logical relations model for the previously mentioned contextual equivalence. We show how to prove necessary properties of the model in order for it to be used to prove contextual equivalences.
Finally, we prove some contextual equivalences in System F, using the logical relations model.

Mathias Pedersen,
Aarhus, January 2022.

Contents

Chapter 1

Introduction

It is often useful to have a notion of program equivalence. For example, after a structural refactoring, the resulting program should be equivalent to the previous one. You may also have two implementations of an algorithm or data structure, one of which is slower, but simpler, so that you are confident about its correctness, while the other is more efficient, but also more complex. Knowing that the two implementations are equivalent allows you to use the more efficient one without worrying about correctness.

One notion of program equivalence is *Contextual Equivalence*, which will be one of the main focus points of this report. Informally, contextual equivalence states that we consider two programs equivalent, if, when used as a part of an arbitrary, bigger program, each could be used in place of the other without changing the behaviour of the whole program.

We will need a language on which we can define and use contextual equivalence. In this report, we will be using System F, which is the simply typed lambda calculus with parametric polymorphism. In chapter 2 we shall define the language formally and discuss some properties of it, which will help us get to know it, and will also be useful in later proofs.

Then, in chapter 3, we shall give two equivalent definitions of contextual equivalence for System F. The first definition reflects the informal explanation just given, and the second gives deeper insight into the nature of contextual equivalence.

It turns out that working directly with contextual equivalence is quite troublesome, so to help us prove equivalences, we shall employ *logical relations*. Logical relations is a general proof technique with many applications in programming language theory, one of which is proving contextual equivalences. In chapter 4 we define a logical relations model, and show some required properties of it, which then allows us to use it for proving contextual equivalences.

Finally, in chapter 5, we shall show and prove some interesting contextual equivalences using the logical relations model. Among these are commutativity of evaluation of pairs and lambda hoisting.

Chapter 2

Definition of Language

To talk about contextual equivalence of programs, we first need to define a language, which captures what programs can be. Consequently, the choice of language influences the definition of contextual equivalence. We will see specifically how this manifests when we reach chapter 3. In this chapter, we will focus on formally defining the language, and showing some facts about it, which will be useful in later chapters.

2.1 The Language

We define the language in three parts: the syntax, the typing rules, and finally the semantics. The language we will be working with is generally refereed to as *System F*, and we shall do so here as well. However, the features included in this language may be different from other presentations. We note here that we will be working with a Curry-style language, as opposed to a Church-style language. This essentially just means that types are not an intrinsic part of the semantics. Instead, types become a property that programs can have.

2.1.1 Syntax

The syntax captures exactly all programs that can be written in our language – in other words, it governs the "forms" our programs can take. We define it using a context-free grammar with the production rules in Backus-Naur form:

$$
\begin{aligned}
e ::= {} & () \mid & \text{(unit value)} \\
& x \mid & \text{(variables)} \\
& \overline{n} \mid e + e \mid e - e \mid e \leq e \mid e < e \mid e = e \mid & \text{(integers)} \\
& \mathsf{true} \mid \mathsf{false} \mid \mathsf{if}\ e\ \mathsf{then}\ e\ \mathsf{else}\ e \mid & \text{(booleans)} \\
& (e, e) \mid \mathsf{fst}\ e \mid \mathsf{snd}\ e \mid & \text{(products)} \\
& \mathsf{inj}_1\ e \mid \mathsf{inj}_2\ e \mid \mathsf{match}\ e\ \mathsf{with}\ \mathsf{inj}_1\ x \Rightarrow e \mid \mathsf{inj}_2\ x \Rightarrow e\ \mathsf{end} \mid & \text{(sums)} \\
& \lambda x.\ e \mid e\ e & \text{(functions)} \\
& \Lambda\ e \mid e\ \_ & \text{(polymorphism)}
\end{aligned}
$$

All programs in our language can be derived using the rules above. We shall also define the syntax of values, types, and evaluation contexts in a similar fashion.

$$v ::= () \mid \overline{n} \mid \mathsf{true} \mid \mathsf{false} \mid (v, v) \mid \mathsf{inj}_1\ v \mid \mathsf{inj}_2\ v \mid \lambda x.\ e \mid \Lambda\ e \qquad \text{(values)}$$

$$\tau ::= \mathsf{Unit} \mid \mathbb{Z} \mid \mathbb{B} \mid \tau \times \tau \mid \tau + \tau \mid \tau \to \tau \mid \forall X.\ \tau \qquad \text{(types)}$$

$$\begin{aligned} K ::= & [] \mid K + e \mid v + K \mid K - e \mid v - K \mid K \leq e \mid & \text{(evaluation context)} \\ & v \leq K \mid K < e \mid v < K \mid K = e \mid v = K \mid \\ & \mathsf{if}\ K\ \mathsf{then}\ e\ \mathsf{else}\ e \mid (K, e) \mid (v, K) \mid \mathsf{fst}\ K \mid \mathsf{snd}\ K \mid \\ & \mathsf{inj}_1\ K \mid \mathsf{inj}_2\ K \mid \\ & \mathsf{match}\ K\ \mathsf{with}\ \mathsf{inj}_1\ x \Rightarrow e \mid \mathsf{inj}_2\ x \Rightarrow e\ \mathsf{end} \mid \\ & K\ e \mid v\ K \mid K\ \_ \end{aligned}$$

Evaluation contexts will be used when we get to defining the semantics. In general, contexts like this are simply programs with a "hole". That hole can be plugged by some expressions e, and in that case we write $K[e]$. We will see another example of a context in chapter 3.
Note that programs in our language may not necessarily make "sense". For example, using the rules above, we can construct the program $\mathsf{true} + 1$. To avert this problem, we introduce the typing rules, which will allow us to only talk about well-typed programs.

2.1.2 Typing Rules

The typing rules give us a way to derive the type of a program in our language. Before defining them, we need to introduce a few concepts.
Since we have type abstractions, we need a way to keep track of introduced type variables. The type environment, Ξ, achieves this for us, by simply being a set containing type-variables. Likewise, given we have functions, we need to keep track of the types of introduced variables. Here, the variable environment, Γ, functions like a map, telling us which type a variable is bound to. Note that any type-variables mentioned by Γ must be present in Ξ for it to be a valid variable environment when used in a judgement. A judgement, $\Xi \mid \Gamma \vdash e : \tau$, states that e has type τ under Ξ and Γ.
With this in place, we are now ready to present the typing rules. They are

defined using inference rules as follows.

$$\textsc{T-Var}\ \frac{(x:\tau)\in\Gamma}{\Xi\mid\Gamma\vdash x:\tau}\qquad \textsc{T-Unit}\ \frac{}{\Xi\mid\Gamma\vdash ():\mathsf{Unit}}\qquad \textsc{T-Int}\ \frac{}{\Xi\mid\Gamma\vdash\overline{n}:\mathbb{Z}}$$

$$\textsc{T-Add}\ \frac{\Xi\mid\Gamma\vdash e_1:\mathbb{Z}\qquad \Xi\mid\Gamma\vdash e_2:\mathbb{Z}}{\Xi\mid\Gamma\vdash e_1+e_2:\mathbb{Z}}\qquad \textsc{T-Sub}\ \frac{\Xi\mid\Gamma\vdash e_1:\mathbb{Z}\qquad \Xi\mid\Gamma\vdash e_2:\mathbb{Z}}{\Xi\mid\Gamma\vdash e_1-e_2:\mathbb{Z}}$$

$$\textsc{T-Le}\ \frac{\Xi\mid\Gamma\vdash e_1:\mathbb{Z}\qquad \Xi\mid\Gamma\vdash e_2:\mathbb{Z}}{\Xi\mid\Gamma\vdash e_1\le e_2:\mathbb{B}}\qquad \textsc{T-Lt}\ \frac{\Xi\mid\Gamma\vdash e_1:\mathbb{Z}\qquad \Xi\mid\Gamma\vdash e_2:\mathbb{Z}}{\Xi\mid\Gamma\vdash e_1< e_2:\mathbb{B}}$$

$$\textsc{T-Eq}\ \frac{\Xi\mid\Gamma\vdash e_1:\mathbb{Z}\qquad \Xi\mid\Gamma\vdash e_2:\mathbb{Z}}{\Xi\mid\Gamma\vdash e_1= e_2:\mathbb{B}}\qquad \textsc{T-True}\ \frac{}{\Xi\mid\Gamma\vdash \mathsf{true}:\mathbb{B}}\qquad \textsc{T-False}\ \frac{}{\Xi\mid\Gamma\vdash \mathsf{false}:\mathbb{B}}$$

$$\textsc{T-If}\ \frac{\Xi\mid\Gamma\vdash e_1:\mathbb{B}\qquad \Xi\mid\Gamma\vdash e_2:\tau\qquad \Xi\mid\Gamma\vdash e_3:\tau}{\Xi\mid\Gamma\vdash \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3:\tau}$$

$$\textsc{T-Pair}\ \frac{\Xi\mid\Gamma\vdash e_1:\tau_1\qquad \Xi\mid\Gamma\vdash e_2:\tau_2}{\Xi\mid\Gamma\vdash (e_1,e_2):\tau_1\times\tau_2}\qquad \textsc{T-Fst}\ \frac{\Xi\mid\Gamma\vdash e:\tau_1\times\tau_2}{\Xi\mid\Gamma\vdash \mathsf{fst}\ e:\tau_1}$$

$$\textsc{T-Snd}\ \frac{\Xi\mid\Gamma\vdash e:\tau_1\times\tau_2}{\Xi\mid\Gamma\vdash \mathsf{snd}\ e:\tau_2}$$

$$\textsc{T-Inj1}\ \frac{\Xi\mid\Gamma\vdash e:\tau_1}{\Xi\mid\Gamma\vdash \mathsf{inj}_1\ e:\tau_1+\tau_2}\qquad \textsc{T-Inj2}\ \frac{\Xi\mid\Gamma\vdash e:\tau_2}{\Xi\mid\Gamma\vdash \mathsf{inj}_2\ e:\tau_1+\tau_2}$$

$$\textsc{T-Match}\ \frac{\Xi\mid\Gamma\vdash e:\tau_1+\tau_2\qquad \Xi\mid\Gamma,x:\tau_1\vdash e_2:\tau\qquad \Xi\mid\Gamma,x:\tau_2\vdash e_3:\tau}{\Xi\mid\Gamma\vdash \mathsf{match}\ e_1\ \mathsf{with}\ \mathsf{inj}_1\ x\Rightarrow e_2\mid \mathsf{inj}_2\ x\Rightarrow e_3\ \mathsf{end}:\tau}$$

$$\textsc{T-Lam}\ \frac{\Xi\mid\Gamma,x:\tau_1\vdash e:\tau_2}{\Xi\mid\Gamma\vdash \lambda x.\ e:\tau_1\to\tau_2}\qquad \textsc{T-App}\ \frac{\Xi\mid\Gamma\vdash e_1:\tau_1\to\tau_2\qquad \Xi\mid\Gamma\vdash e_2:\tau_1}{\Xi\mid\Gamma\vdash e_1\ e_2:\tau_2}$$

$$\textsc{T-TLam}\ \frac{\Xi,X\mid\Gamma\vdash e:\tau}{\Xi\mid\Gamma\vdash \Lambda\, e:\forall X.\ \tau}\qquad \textsc{T-TApp}\ \frac{\Xi\mid\Gamma\vdash e:\forall X.\ \tau}{\Xi\mid\Gamma\vdash e\ \_:\tau[\tau'/X]}$$

2.1.3 Semantics

Finally, we define the semantics of the language. We do this using a single "step" rule, and several "head-step" rules. The head-step rules govern all possible reductions, and the "head-step-step" rule tells us where we may apply these reductions.

$$\textsc{head-step-step}\quad \frac{e \rightarrow_h e'}{K[e] \rightarrow K[e']}$$

The evaluation context, K, is the part that tells us where in our program we can apply a reduction. If we have an expression e, and we have $e = K[f]$, and $f \rightarrow_h f'$, then we may conclude that $e \rightarrow K[f']$. Now, the head-steps are as follows.

$$\textsc{E-add}\quad \overline{n_1} + \overline{n_2} \rightarrow_h \overline{n_1 + n_2}$$

$$\textsc{E-sub}\quad \overline{n_1} - \overline{n_2} \rightarrow_h \overline{n_1 - n_2}$$

$$\textsc{E-eq}\quad \frac{n_1 = n_2}{\overline{n_1} = \overline{n_2} \rightarrow_h \mathsf{true}}$$

$$\textsc{E-not-eq}\quad \frac{n_1 \neq n_2}{\overline{n_1} = \overline{n_2} \rightarrow_h \mathsf{false}}$$

$$\textsc{E-le}\quad \frac{n_1 \leq n_2}{\overline{n_1} \leq \overline{n_2} \rightarrow_h \mathsf{true}}$$

$$\textsc{E-not-le}\quad \frac{n_1 \not\leq n_2}{\overline{n_1} \leq \overline{n_2} \rightarrow_h \mathsf{false}}$$

$$\textsc{E-lt}\quad \frac{n_1 < n_2}{\overline{n_1} < \overline{n_2} \rightarrow_h \mathsf{true}}$$

$$\textsc{E-not-lt}\quad \frac{n_1 \not< n_2}{\overline{n_1} < \overline{n_2} \rightarrow_h \mathsf{false}}$$

$$\textsc{E-if-true}\quad \mathsf{if\ true\ then}\ e_2\ \mathsf{else}\ e_3 \rightarrow_h e_2$$

$$\textsc{E-if-false}\quad \mathsf{if\ false\ then}\ e_2\ \mathsf{else}\ e_3 \rightarrow_h e_3$$

$$\textsc{E-fst}\quad \mathsf{fst}\ (v_1, v_2) \rightarrow_h v_1$$

$$\textsc{E-snd}\quad \mathsf{snd}\ (v_1, v_2) \rightarrow_h v_2$$

$$\textsc{E-match-inj1}\quad \mathsf{match}\ (\mathsf{inj}_1\ v)\ \mathsf{with\ inj}_1\ x \Rightarrow e_2 \mid \mathsf{inj}_2\ x \Rightarrow e_3\ \mathsf{end} \rightarrow_h e_2[v/x]$$

$$\textsc{E-match-inj2}\quad \mathsf{match}\ (\mathsf{inj}_2\ v)\ \mathsf{with\ inj}_1\ x \Rightarrow e_2 \mid \mathsf{inj}_2\ x \Rightarrow e_3\ \mathsf{end} \rightarrow_h e_3[v/x]$$

$$\textsc{E-lam-app}\quad (\lambda x.\ e)\ v \rightarrow_h e[v/x]$$

$$\textsc{E-tapp-tlam}\quad (\Lambda\ e)\ \_ \rightarrow_h e$$

Notice that we are working with a call-by-value language in that a β-reduction may only happen when the argument is a value. A substitution $e[e'/x]$ is read as e with all occurrences of x substituted with e'. We will not define formally how a substitution $e[e'/x]$ works here, but refer instead to section 5.3 in [1], which explains how to formally define substitutions so that they are *capture-avoiding*.

To strike home the workings of evaluation context, we give here an example. Let's say we want to show that $(\lambda x.\ x)\ 2 + 3 \rightarrow (\lambda x.\ x)\ 5$. Using the evaluation context $K = (\lambda x.\ x)\ []$, which is a valid evaluation context, we may conclude using the head-step-step rule that $K[2+3] \rightarrow K[5]$, if $2+3 \rightarrow_h 5$. By the E-add head-step rule, we know that $2 + 3 \rightarrow_h 5$, thus we have that $(\lambda x.\ x)\ 2 + 3 \equiv$

$K[2+3] \rightarrow K[5] \equiv (\lambda x.\ x)\ 5$. Further, using $K = []$ and the headstep rule E-lam-app, we may conclude $(\lambda x.\ x)\ 5 \rightarrow x[5/x] \equiv 5$. We can hide the middle step, and simply write $(\lambda x.\ x)\ 2+3 \rightarrow^2 5$, meaning that we perform two steps to reach 5. We also introduce the notation $e \rightarrow^* e'$, which means that e takes zero or more steps to reach e'. And if $e \rightarrow^* v$, then we shall write $e \Downarrow v$.
In the majority of the report, we will leave the evaluation contexts implicit and simply refer to the head-steps when justifying reductions.

At this point, our language has been formally defined, so next we will note down some facts and results about it.

2.2 Properties of the Language

In this section we note down some useful properties of our language, which will help us reason about the concepts in later chapters. First we show some interesting lemmas about the evaluation context.

2.2.1 Evaluation Context

Lemma 1. $K[e] \rightarrow_h e' \wedge \neg(K = []) \implies \mathrm{Val}(e)$

Proof. This can be shown by doing case distinction on the head-step $K[e] \rightarrow_h e'$. We show it here only for case E-ADD, as all the other cases are similar.

So assume $K[e] = \overline{n_1} + \overline{n_2}$, and $e' = \overline{n_1 + n_2}$. Then there are three cases: $K = []$ and $e = \overline{n_1} + \overline{n_2}$, $K = [] + \overline{n_2}$ and $e = \overline{n_1}$, or $K = \overline{n_1} + []$ and $e = \overline{n_2}$. The first case raises a contradiction as we have assumed $\neg(K = [])$. In the remaining two cases, we may conclude $\mathrm{Val}(e)$, as wanted. □

Lemma 2 (Evaluation under Context)**.** $K[e] \rightarrow^* e' \implies \exists e''.(e \rightarrow^* e'') \wedge ((\mathrm{Val}(e'') \wedge K[e''] \rightarrow^* e') \vee (\neg\mathrm{Val}(e'') \wedge K[e''] = e'))$

Proof. So assuming $K[e] \rightarrow^* e'$, we must show

$$\exists e''.(e \rightarrow^* e'') \wedge \big((\mathrm{Val}(e'') \wedge K[e''] \rightarrow^* e') \vee (\neg\mathrm{Val}(e'') \wedge K[e''] = e')\big) \tag{2.1}$$

We proceed by induction on the number of steps in the evaluation $K[e] \rightarrow^* e'$. Let n denote the number of steps taken, so that $K[e] \rightarrow^n e'$.

B.C. $n = 0$. In this case, we have that $K[e] \rightarrow^0 e'$, which means that $K[e] = e'$. Now use e for e'' in 2.1. We must show

$$(e \rightarrow^* e) \wedge \big((\mathrm{Val}(e) \wedge K[e] \rightarrow^* e') \vee (\neg\mathrm{Val}(e) \wedge K[e] = e')\big)$$

Trivially, $e \rightarrow^* e$. For the second part, we proceed by case distinction on $\mathrm{Val}(e)$.

case $\mathrm{Val}(e)$. We have that $K[e] \to^0 e'$, so $K[e] \to^* e'$. Thus, we have $\mathrm{Val}(e) \wedge K[e] \to^* e'$, which matches the left part of the "$\vee$".

case $\neg\mathrm{Val}(e)$. We know that $K[e] = e'$, so we have $\neg\mathrm{Val}(e) \wedge K[e] = e'$, which matches the right part of the "$\vee$".

I.S. $n = m + 1$. Now we have $K[e] \to^{m+1} e'$. The induction hypothesis is:

$$\forall F, f, f'. F[f] \to^m f' \implies \exists f''.(f \to^* f'')\wedge \\ \big((\mathrm{Val}(f'') \wedge F[f''] \to^* f') \vee (\neg\mathrm{Val}(f'') \wedge K[f''] = f')\big) \quad (2.2)$$

Split the evaluation $K[e] \to^{m+1} e'$ up, so that $K[e] \to g \wedge g \to^m e'$. Looking at our semantics, we must have that $K[e] = H[h]$, and $g = H[h']$, for some evaluation context H, and expressions h, h', and $h \to_h h'$. There are now three possible cases. Either $e = h$, e is a superexpression of h, or e is a subexpression of h. We will consider each in turn.

case $K = H$ and $e = h$. Then $g = H[h'] = K[h']$, and $e \to_h h'$. Furthermore, $K[h'] \to^m e'$. Instantiate I.H. with this to get

$$\exists f''.(h' \to^* f'')\wedge \\ \big((\mathrm{Val}(f'') \wedge K[f''] \to^* e') \vee (\neg\mathrm{Val}(f'') \wedge K[f''] = e')\big) \quad (2.3)$$

Now use f'' for e'' in 2.1. We must then show

$$(e \to^* f'') \wedge \big((\mathrm{Val}(f'') \wedge K[f''] \to^* e') \vee (\neg\mathrm{Val}(f'') \wedge K[f''] = e')\big)$$

We know that $e \to h'$, as $e \to_h h'$, and by 2.3, we know that $h' \to^* f''$, so that $e \to^* f''$. The second part follows directly from 2.3.

case $K[E[]] = H$ and $e = E[h]$. Then $g = H[h'] = K[E[]][h'] = K[E[h']]$, thus $K[E[h']] \to^m e'$. Instantiate I.H. with this to get

$$\exists f''.(E[h'] \to^* f'')\wedge \\ \big((\mathrm{Val}(f'') \wedge K[f''] \to^* e') \vee (\neg\mathrm{Val}(f'') \wedge K[f''] = e')\big) \quad (2.4)$$

Using f'' for e'' in 2.1, we must show

$$(e \to^* f'') \wedge \big((\mathrm{Val}(f'') \wedge K[f''] \to^* e') \vee (\neg\mathrm{Val}(f'') \wedge K[f''] = e')\big)$$

We know that $E[h] \to E[h']$, as $h \to_h h'$, and since $e = E[h]$, then $e \to E[h']$. By 2.4, we know that $E[h'] \to^* f''$, so that $e \to^* f''$. The second part follows directly from 2.4.

case $K = H[E[]]$ and $E[e] = h$. Here we have $h = E[e]$, so $E[e] \to_h h'$. Note that E is not the empty evaluation context as, otherwise, we would be in the first case. So by lemma 1, we know that $\mathrm{Val}(e)$. Now, pick e for e'' in 2.1. We must show

$$(e \to^* e) \wedge \big((\mathrm{Val}(e) \wedge K[e] \to^* e') \vee (\neg\mathrm{Val}(e) \wedge K[e] = e')\big)$$

Trivially, $e \to^* e$. We also have that $\mathrm{Val}(e)$, and since $K[e] \to^{m+1} e'$, then $K[e] \to^* e'$.

□

A straightforward result from this lemma is the following corollary.

Corollary 2.1. $K[e] \to^* v \implies \exists v'.(e \to^* v') \wedge K[v'] \to^* v$

2.2.2 Substitution

We will be needing a way to close off open expressions later, so we shall define here an extended substitution $\gamma = \{x_1 \mapsto v_1, x_2 \mapsto v_2, \dots\}$ inductively as follows.

$$\emptyset(e) = e$$
$$\gamma[x \mapsto v](e) = \gamma(e[v/x])$$

We require of γ that all values v_i are closed and all variables x_i are distinct. The following lemma will be useful in later proofs.

Lemma 3 (Substitution). $\gamma[x \mapsto v](e) = \gamma(e)[v/x]$

Proof. Similar to Lemma 10 in [2]. □

2.2.3 Type Safety, Normalisation, and Determinacy

We note here a few interesting results about System F. They will not all be proved here, as it requires a substantial amount of work.
Firstly, System F is a type-safe language. Essentially, this means that well-typed programs don't get "stuck" in the sense that either we can take a step, or we have reduced to a value. We may state this formally as follows.

Theorem 4. $\bullet \mid \bullet \vdash e : \tau \wedge e \to^* e' \implies \mathrm{Val}(e') \vee \exists e''.e' \to e''$

Secondly, every closed and well-typed expression in System F has a reduction to normal form. An expression is in normal form if it cannot be reduced any further. This property is called "normalisation".

Theorem 5. *System F is a normalising language.*

Together with type-safety, normalisation implies that all closed and well-typed expressions will terminate at some value. We will actually prove this theorem later using the logical relations model we define in chapter 4. As an aside, there is a stronger version of normalisation called "strong normalisation". This essentially just states the above theorem holds for *all* reductions of every closed and well-typed expression. System F is strongly normalising, but this is no surprise given the next property of System F we will see.

Finally, System F is a deterministic language.

Theorem 6. *If* $e \to e_1$ *and* $e \to e_2$ *then* $e_1 = e_2$

A consequence of this is the following corollary.

Corollary 6.1. $e \Downarrow v_1 \wedge e \Downarrow v_2 \implies v_1 = v_2$

Proof. We proceed by induction on the number of steps of the evaluation $e \Downarrow v_1$.

B.C $n = 0$. Since $e \to^0 v_1$, then $e = v_1$, meaning that $v_1 \Downarrow v_2$, which implies that $v_1 = v_2$.

I.S. $n = m + 1$. Our induction hypothesis is $\forall e', v_1', v_2'.e' \to^m v_1' \wedge e' \Downarrow v_2' \implies v_1' = v_2'$. Now, in this case we have that $e \to^{m+1} v_1$. We split this reduction up as follows: $e \to e_1$ and $e_1 \to^m v_1$, for some e_1. Note that this implies that e is not a value. Thus, we may conclude that the evaluation, $e \to^* v_2$, uses at least one step. We split this up as well so that $e \to e_2$ and $e_2 \to^* v_2$, for some e_2. By determinacy (theorem 6), we know that $e_1 = e_2$. Thus $e_1 \to^* v_2$. We may now apply the induction hypothesis, using e_1 for e', v_1 for v_1', and v_2 for v_2', and conclude that $v_1 = v_2$.

□

Chapter 3

Contextual Equivalence

Imagine you are writing a program, and as part of that program, you need to use a stack. So as part of your program, you implement a stack. However, your implementation is naive, and not efficient. Thus you implement a new stack, that is more complex, but also more efficient. You would of course want to use the more efficient stack implementation in your program, but since it is more complex, you are not sure whether you can justify the refactoring – the implementations might show differing behaviour.
What you want to show, then, is that your two stack implementations *behave* the same. In other words, no matter which *context* you use your two stack implementations in, they will always behave the same as part of the context. This is intuitively what we want to define with *Contextual Equivalence*.
Contextual Equivalence can be used to show this along with other, "smaller" properties such as idempotency and hoisting of functions. Both of which we will be proving in chapter 5.
In this chapter we give two equivalent definitions of what it means for programs to be contextually equivalent. The first definition will be more intuitive and instructive, while the second will give a deeper insight into the theoretical underpinnings of contextual equivalence.

3.1 Definition of Contextual Equivalence

In order to define contextual equivalence, we will first need to define the notion of a context. As with the evaluation context, this will simply be a program with exactly one hole in it, which may be plugged by some expression.

Definition 3.1.1 (Context)**.** A context C is anything that may be derived using the following CFG.

$$
\begin{aligned}
C ::= &[\cdot] \mid C + e \mid e + C \mid C - e \mid e - C \mid C \leq e \mid e \leq C \mid C < e \mid e < C \mid \\
&C = e \mid e = C \mid \textsf{if } C \textsf{ then } e \textsf{ else } e \mid \textsf{if } e \textsf{ then } C \textsf{ else } e \mid \textsf{if } e \textsf{ then } e \textsf{ else } C \mid \\
&(C, e) \mid (e, C) \mid \textsf{fst } C \mid \textsf{snd } C \mid \textsf{inj}_1\ C \mid \textsf{inj}_2\ C \mid \\
&\textsf{match } C \textsf{ with inj}_1\ x \Rightarrow e \mid \textsf{inj}_2\ x \Rightarrow e \textsf{ end} \mid \\
&\textsf{match } e \textsf{ with inj}_1\ x \Rightarrow C \mid \textsf{inj}_2\ x \Rightarrow e \textsf{ end} \mid \\
&\textsf{match } e \textsf{ with inj}_1\ x \Rightarrow e \mid \textsf{inj}_2\ x \Rightarrow C \textsf{ end} \mid \lambda x.\ C \mid C\ e \mid e\ C \mid \Lambda\ C \mid C\ \_
\end{aligned}
$$

Unlike the evaluation context, the hole can here be *anywhere* in the program. This is due to how we will be defining contextual equivalence. Intuitively, we only want to consider two expressions contextually equivalent, when we can plug them into *any* program with a hole, and they exhibit similar behaviour.
Next, we shall define "Context Typing". The type of a context will capture the type of the hole, and the type of the plugged program. We define it as an inference rule:

$$
\text{T-CTX}\quad \frac{\Xi \mid \Gamma \vdash e : \tau \qquad \Xi' \mid \Gamma' \vdash C[e] : \tau'}{C : (\Xi \mid \Gamma \vdash \tau) \Rightarrow (\Xi' \mid \Gamma' \vdash \tau')}
$$

We can read this as: the context C takes an expression e (the expression which plugs the hole) of type τ under Ξ, Γ, and outputs an expression $C[e]$ (the plugged program) of type τ' under Ξ', Γ'.
With this defined, we are now ready to define contextual equivalence. We first state the definition, and then give some intuition as to why this definition makes sense.

Definition 3.1.2 (Contextual Equivalence)**.** We define contextual equivalence of two expressions e_1 and e_2 at type τ under Ξ and Γ as

$$
\begin{gathered}
\Xi \mid \Gamma \vdash e_1 \approx^{ctx} e_2 : \tau \\
\iff \\
\Xi \mid \Gamma \vdash e_1 : \tau \quad \wedge \quad \Xi \mid \Gamma \vdash e_2 : \tau \quad \wedge \\
\forall C : (\Xi \mid \Gamma \vdash \tau) \Rightarrow (\bullet \mid \bullet \vdash \mathbb{B}), v.(C[e_1] \Downarrow v \iff C[e_2] \Downarrow v)
\end{gathered}
$$

This says that two expressions are contextually equivalent when they are well typed and we can plug them into any context, which closes both expressions and makes the plugged programs have type $\mathbb{B}$, and one program, $C[e_1]$, terminates at some Boolean v (thus, either $\textsf{true}$ or $\textsf{false}$) if and only if the other program, $C[e_2]$, terminates at the same Boolean, v.
To help demystify this definition, we will discuss a few points. Firstly, why we have decided to use $\mathbb{B}$ as the type of the plugged programs; there are plenty other types that could have been used instead such as $\textsf{Unit}$ or $\mathbb{Z}$. Why not use those? The answer is that using $\mathbb{B}$ is "strong enough". We will discuss what this means in a moment. But we could in fact have used some other type like $\mathbb{Z}$ or

even a composite type like $\mathbb{B} \times \mathbb{Z}$. The important part is that we are able to make the plugged programs exhibit different behaviour. We cannot do this if the plugged programs have type Unit; by theorem 4 and 5 we know that both plugged programs terminate at a value, and there is only one value with type Unit, namely $()$, so both programs will terminate at $()$ regardless of what e_1 and e_2 looks like. Had we been dealing with a non-terminating language, then we could have used Unit, since it would then have two possible differing behaviours: terminate with $()$ and never terminate. In that case, we would define contextual equivalence as $C[e_1] \Downarrow \iff C[e_2] \Downarrow$.

Now, let's discuss why our definition is strong enough, and what that means. Firstly, when talking about contextual equivalence, we are not interested in whether or not expressions are syntactically the same – we only care about their observable behaviour. So let's say that two contextually equivalent expressions e_1 and e_2 run down to some values, then those values should be behaviourally the same. We have not yet defined formally what we mean by "behaviourally the same". However, intuitively, if the expressions are of integer type, for example, then the two values should be the same number. And for product types, the first value in each pair should be behaviourally the same, and likewise with the second value in each pair. Our definition of contextual equivalence ensures this. To see why this is the case, consider two contextually equivalent programs e_1 and e_2, where $\bullet \mid \bullet \vdash e_1 : \tau$ and $\bullet \mid \bullet \vdash e_2 : \tau$. Now consider what happens if e_1 terminates with some value v_1. Can we then guarantee that e_2 also terminates with some value v_2, and v_1 and v_2 behave the same? Since e_1 and e_2 are contextually equivalent, then we can put them into any context, C, and one will terminate at some value v if and only if the other one does. Let's first consider when the expressions are of integer type (so $\tau = \mathbb{Z}$). Then consider when C has the form $[\cdot] = v_1$. Here $C[e_1] \Downarrow \mathsf{true}$. But what about $C[e_2]$? If $v_2 \neq v_1$, then our reduction rule E-not-eq tells us that $C[e_2] \Downarrow \mathsf{false}$. However, since e_1 and e_2 are contextually equivalent, and $C[e_1]$ reduces to true, then we know that $C[e_2]$ must reduce to true. Hence it is not the case that $v_2 \neq v_1$, thus $v_2 = v_1$. So if our two programs of type integer are contextually equivalent, then they must both evaluate to the same number.

It becomes a little more difficult to reason about when τ is not a base-type, like Unit, $\mathbb{Z}$, or $\mathbb{B}$. Take function type, for instance. If e_1 and e_2 both evaluate down to functions, how do we know that those functions are behaviourally equivalent? To see this, consider the following "congruence" rule:

$$\textsc{Cng-ctx-app}\quad \frac{\Xi \mid \Gamma \vdash f \approx^{ctx} f' : \tau \to \tau' \qquad \Xi \mid \Gamma \vdash t \approx^{ctx} t' : \tau}{\Xi \mid \Gamma \vdash f\ t \approx^{ctx} f'\ t' : \tau'}$$

This essentially gives us what we want: if we have two contextually equivalent functions, then, as long as we give them contextually equivalent inputs, the outputs will also be contextually equivalent. That output may be another function, but then this rule applies again to that output, and so on. In other words, we can keep on applying this rule until we at some point get a base type, like $\mathbb{Z}$, and at that point, we know that the two numbers will be the same, by the argument made above.

Of course, it may happen that the output is of another non-base type, such as type abstraction. But we may state similar congruence rules for all other non-base types. However proving all of them is quite tedious, so we will here only prove the congruence rule for function application.

Proof. By the two hypotheses of the rule, we know that $\Xi \mid \Gamma \vdash f : \tau \to \tau'$ and $\Xi \mid \Gamma \vdash t : \tau$. So by the T-APP rule, we may conclude $\Xi \mid \Gamma \vdash f\ t : \tau'$. Likewise for the prime variants. So all that remains to be shown is that $\forall C : (\Xi \mid \Gamma \vdash \tau') \Rightarrow (\bullet \mid \bullet \vdash \mathbb{B}), v.(C[f\ \ t] \Downarrow v \iff C[f'\ \ t'] \Downarrow v)$. So assume some context, C, of the right type, and some value v. Consider now the context $C[[\cdot]\ t]$. We know that f is contextually equivalent to f', so $\forall v'.C[f\ t] \Downarrow v' \iff C[f'\ t] \Downarrow v'$. Now consider the context $C[f'\ [\cdot]]$. Since t is contextually equivalent to t', then we know $\forall v''.C[f'\ t] \Downarrow v'' \iff C[f'\ t'] \Downarrow v''$. Now it follows by instantiating both of the prior results with v. Then we have $C[f\ t] \Downarrow v \iff C[f'\ t] \Downarrow v \iff C[f'\ t'] \Downarrow v$, which was what we wanted. □

The way we have defined contextual equivalence is quite instructive and intuitive. However, one may define contextual equivalence in another, equivalent way, which gives much insight into contextual equivalence, and will make working with the logical relations model later more intuitive. We will explore this alternative definition in the next section.

3.2 Alternative Definition of Contextual Equivalence

Before giving the alternative definition, we first introduce a few concepts. In the following, we will be working with relations $R \subseteq \text{TENV} \times \text{VENV} \times \text{EXPR} \times \text{EXPR} \times \text{TYPE}$, and R will refer to any such relation. TENV is simply the set of all type environments, so $\Xi \in \text{TENV}$. Likewise with the other sets. We also introduce the following notation: $R(\Xi, \Gamma, e, e', \tau) \triangleq \Xi \mid \Gamma \vdash e \approx e' : \tau$, and use them interchangeably.
Now we define two properties that a relation R may have.

Definition 3.2.1 (Adequacy)**.** We say R is an adequate relation when it holds for R that $\bullet \mid \bullet \vdash e \approx e' : \mathbb{B} \implies e \Downarrow v \iff e' \Downarrow v$.

Definition 3.2.2 (Congruency)**.** We say R is a congruence relation (with respect to the typing rules) when it satisfies all the congruence rules that arises from the typing rules.

For instance, the congruence rules for T-unit and T-app would be:

$$\textsc{Cng-unit}\quad \frac{}{\Xi \mid \Gamma \vdash () \approx () : \mathsf{Unit}} \qquad\qquad \textsc{Cng-Tapp}\quad \frac{\Xi \mid \Gamma \vdash e \approx e' : \forall X.\, \tau}{\Xi \mid \Gamma \vdash e\ \_ \approx e'\ \_ : \tau[\tau'/X]}$$

The remaining congruence rules can be found in appendix A.
Now we may define contextual equivalence in the following way.

Definition 3.2.3. Contextual Equivalence is a relation, $CE \subseteq$ TENV $\times$ VENV $\times$ EXPR $\times$ EXPR $\times$ TYPE, such that

- all expressions in CE are well-typed
- CE is a congruence relation
- CE is an adequate relation

Finally, it is the coarsest such relation.

CE being the coarsest such relation means that if given a relation R satisfying the three properties, then $\forall \Xi, \Gamma, e, e', \tau . R(\Xi, \Gamma, e, e', \tau) \implies CE(\Xi, \Gamma, e, e', \tau)$. We of course have to show that this definition of Contextual Equivalence is actually equivalent to the one given in definition 3.1.2:

Theorem 7. $\Xi \mid \Gamma \vdash e \approx^{ctx} e' : \tau \iff CE(\Xi, \Gamma, e, e', \tau)$

Proof. Simply follows from theorems 8 and 11 below. □

Theorem 8 corresponds to showing the " $\implies$ " direction of theorem 7.

Theorem 8. $\cdot \mid \cdot \vdash \cdot \approx^{ctx} \cdot : \cdot \subseteq$ TENV $\times$ VENV $\times$ EXPR $\times$ EXPR $\times$ TYPE *is a congruence relation, an adequate relation, all expressions in the relation are well-typed, and hence by definition of CE, it is included in CE.*

Proof. First, the well-typedness follows directly from the definition of contextual equivalence.
Second, let's show that it is an adequate relation. So assuming $\bullet \mid \bullet \vdash e \approx^{ctx} e' : \mathbb{B}$, we must show $e \Downarrow v \iff e' \Downarrow v$. We know that $\forall C : (\bullet \mid \bullet \vdash \mathbb{B}) \Rightarrow (\bullet \mid \bullet \vdash \mathbb{B}), v'.(C[e] \Downarrow v' \iff C[e'] \Downarrow v')$. So specifically for C being the empty context and v' being v, we get $e \Downarrow v \iff e' \Downarrow v$, which was what we wanted.
To show that it is a congruence relation, we must go through all the congruence rules, and show that they hold. We did the one for function application above, Cng-ctx-app. The rest will not be shown here. □

To prove theorem 11, we will need to show two lemmas.

Lemma 9 (Reflexivity)**.** *Let R be a congruence relation. Then* $\Xi \mid \Gamma \vdash e : \tau \implies R(\Xi, \Gamma, e, e, \tau)$.

Proof. By induction on the typing derivation of e. It follows immediately from applying the induction hypothesis and applying congruency rules. We show one case here.

case T-tlam.
Assuming $\Xi \mid \Gamma \vdash \Lambda\ e : \forall X.\ \tau$ and $\Xi, X \mid \Gamma \vdash e : \tau$, we must show $R(\Xi, \Gamma, \Lambda\ e, \Lambda\ e, \forall X.\ \tau)$. Our induction hypothesis is $R((\Xi, X), \Gamma, e, e, \tau)$. By the congruency rule Cng-tlam, we then get $\Xi \mid \Gamma \vdash \Lambda\ e \approx \Lambda\ e : \forall X.\ \tau$, which was what we wanted.

□

Lemma 10. *Let R be a congruence relation, then* $R(\Xi, \Gamma, e, e', \tau) \wedge C : (\Xi \mid \Gamma \vdash \tau \Rightarrow \Xi' \mid \Gamma' \vdash \tau') \implies R(\Xi', \Gamma', C[e], C[e'], \tau')$.

Proof. By induction on C. We show here only a few interesting cases. For each non-base-case we show, the induction hypothesis will be $\forall \Xi, \Gamma, \Xi', \Gamma', e, e', \tau, \tau'$. $R(\Xi, \Gamma, e, e', \tau) \wedge C' : (\Xi \mid \Gamma \vdash \tau \Rightarrow \Xi' \mid \Gamma' \vdash \tau') \implies R(\Xi', \Gamma', C'[e], C'[e'], \tau')$, where C' is structurally smaller than C.

case $C = [\cdot]$
: This case is trivial as $[\cdot][e] = e$ and $[\cdot] : (\Xi \mid \Gamma \vdash \tau \Rightarrow \Xi \mid \Gamma \vdash \tau)$.

case $C = \mathsf{fst}\ C'$
: Then $C : (\Xi \mid \Gamma \vdash \tau \Rightarrow \Xi' \mid \Gamma' \vdash \tau_1)$ and $C' : (\Xi \mid \Gamma \vdash \tau \Rightarrow \Xi' \mid \Gamma' \vdash \tau_1 \times \tau_2)$, so our goal is $R(\Xi', \Gamma', C[e], C[e'], \tau_1)$. By I.H. we get $\Xi' \mid \Gamma' \vdash C'[e] \approx C'[e'] : \tau_1 \times \tau_2$, and by cng-fst with this, we get $\Xi' \mid \Gamma' \vdash \mathsf{fst}\ C'[e] \approx \mathsf{fst}\ C'[e'] : \tau_1 \equiv \Xi' \mid \Gamma' \vdash C[e] \approx C[e'] : \tau_1$.

case $C = \lambda x.\ C'$
: Then $C : (\Xi \mid \Gamma \vdash \tau \Rightarrow \Xi' \mid \Gamma' \vdash \tau_1 \to \tau_2)$ and $C' : (\Xi \mid \Gamma \vdash \tau \Rightarrow \Xi' \mid \Gamma', x : \tau_1 \vdash \tau_2)$, so our goal is $R(\Xi', \Gamma', C[e], C[e'], \tau_1 \to \tau_2)$. By I.H. we get $\Xi' \mid \Gamma', x : \tau_1 \vdash C'[e] \approx C'[e'] : \tau_2$, and by cng-lam with this, we get $\Xi' \mid \Gamma' \vdash \lambda x.\ C'[e] \approx \lambda x.\ C'[e'] : \tau_1 \to \tau_2 \equiv \Xi' \mid \Gamma' \vdash C[e] \approx C[e'] : \tau_1 \to \tau_2$.

case $C = C'\ e''$
: Then $C : (\Xi \mid \Gamma \vdash \tau \Rightarrow \Xi' \mid \Gamma' \vdash \tau_2)$ and $C' : (\Xi \mid \Gamma \vdash \tau \Rightarrow \Xi' \mid \Gamma' \vdash \tau_1 \to \tau_2)$, so our goal is $R(\Xi', \Gamma', C[e], C[e'], \tau_2)$. From the context-typing and T-app, we have $\Xi' \mid \Gamma' \vdash e'' : \tau_1$. By lemma 9, we get $\Xi' \mid \Gamma' \vdash e'' \approx e'' : \tau_1$. Now, by the induction hypothesis, we get $\Xi' \mid \Gamma' \vdash C'[e] \approx C'[e'] : \tau_1 \to \tau_2$. And by cng-app with this, we get $\Xi' \mid \Gamma' \vdash C'[e]\ e'' \approx C'[e']\ e'' : \tau_2 \equiv \Xi' \mid \Gamma' \vdash C[e] \approx C[e] : \tau_2$.

□

With these lemmas proved, we are ready to prove theorem 11, which corresponds to showing the " $\Longleftarrow$ " direction of theorem 7.

Theorem 11. *Let R be a relation satisfying the three properties of definition 3.2.3, then* $R(\Xi, \Gamma, e, e', \tau) \implies \Xi \mid \Gamma \vdash e \approx^{ctx} e' : \tau$.

Proof. So assuming $R(\Xi, \Gamma, e, e', \tau)$, we must show $\Xi \mid \Gamma \vdash e : \tau \ \wedge\ \Xi \mid \Gamma \vdash e' : \tau\ \wedge \forall C : (\Xi \mid \Gamma \vdash \tau) \Rightarrow (\bullet \mid \bullet \vdash \mathbb{B}), v.(C[e] \Downarrow v \iff C[e'] \Downarrow v)$. Again, the well-typedness follows from the fact that all expressions in R are well-typed. So what remains to be shown is that $\forall C : (\Xi \mid \Gamma \vdash \tau) \Rightarrow (\bullet \mid \bullet \vdash \mathbb{B}), v.(C[e] \Downarrow v \iff C[e'] \Downarrow v)$. So assume some context C of the right type and some value v. By lemma 10, we have $R(\bullet, \bullet, C[e], C[e'], \mathbb{B})$. It then follows from adequacy of R that $C[e] \Downarrow v \iff C[e'] \Downarrow v$, which was what we wanted. $\square$

Chapter 4

Logical Relations Model for Contextual Equivalence

4.1 Logical Relations

Before we begin defining our logical relations model (LR model for short), let's briefly discuss what they are.
Logical relations may be described in several ways. Firstly, as a mathematical object, it is, as the name suggests, a *relation*. As we shall see in the next section this relation is type-indexed and inductively defined. Furthermore, the relation may be unary, in which case we usually call it a logical predicate.
Although this description is accurate, it doesn't tell us much about their purpose. So another way to describe logical relations is as a proof method. Some properties are hard to show directly – induction attempts may fail due to the induction hypothesis being too weak. An LR model may solve this problem by taking a different approach. Essentially, an LR model acts as an intermediate step – we design the LR model so that everything in the relation has the property of interest. Then, instead of showing that something has the desired property, we show that it is in our relation. Note that the model may not contain *everything* with the property, so it may not be able to prove everything involving the property. In fact designing an LR model that is powerful enough for what you want to show is non-trivial and depends on what you want to show about the property of interest.
However, a common design-principle worth noting is that the property of interest is preserved by eliminating forms. This design-principle will be very common in the LR model we define.
In our case, the property of interest is contextual equivalence of pairs of programs. So when we define our LR model in the next section, we shall do so as to ensure that if two programs are in our relation, then they are also contextually equivalent. By theorem 11, we may do this by ensuring that our LR model is a congruence relation, an adequate relation, and that all expressions in it are well-typed. Furthermore, the theorems we want to show about our property, contextual equivalence, can be found in chapter 5. Interestingly, even though we define our

LR model to prove theorems involving contextual equivalence, it is not limited to that property. So we shall also see that it is capable of proving theorems unrelated to contextual equivalence.
We keep these points in mind as we move on to the next section, where we shall define an LR model.

4.2 Defining the Logical Relations Model

We will be splitting up the definition of the complete LR model into several smaller steps, where each step increases the generality of the terms we relate. Initially, we shall define the "value interpretation" that states which values are logically related. Following this, we define the "expression interpretation", which talks about relatedness for *closed* expressions. Finally, we shall define the "type environment interpretation" and "variable environment interpretation", which will give us relational substitutions allowing us to close off open expressions, so that we may define our complete LR model on open expressions.

A property that we want to hold for all values in the value interpretation is that they are closed and well-typed, so we shall firstly define a well-typedness relation as follows:

$$\mathcal{W}[\![\tau]\!]_\rho(v_1, v_2) \triangleq \bullet \mid \bullet \vdash v_1 : \rho_1(\tau) \wedge \bullet \mid \bullet \vdash v_2 : \rho_2(\tau)$$

The ρ part will be explained shortly. For now, $\rho_i(\tau)$ is simply a closed type.
Now we are prepared to define the value interpretation. For the sake of brevity, we do not include $\mathcal{W}[\![\tau]\!]_\rho(v_1, v_2)$ explicitly when defining the value interpretation for the various types. Firstly, let's define the value interpretation for base types.

$$\begin{aligned}
\mathcal{V}[\![\mathsf{Unit}]\!]_\rho(v_1, v_2) &\triangleq (v_1, v_2) \in \{((), ())\} \\
\mathcal{V}[\![\mathbb{Z}]\!]_\rho(v_1, v_2) &\triangleq (v_1, v_2) \in \{(\overline{n}, \overline{n}) \mid \overline{n} \in \mathbb{Z}\} \\
\mathcal{V}[\![\mathbb{B}]\!]_\rho(v_1, v_2) &\triangleq (v_1, v_2) \in \{(\mathsf{true}, \mathsf{true}), (\mathsf{false}, \mathsf{false})\}
\end{aligned}$$

Remember that we are trying to define an LR model for contextual equivalence. So deciding whether two values of some type should be in our logical relation, we shall keep in mind that they should then also be contextually equivalent. So for the base types, two values are in our relation only when they are the same value. Next, we consider product-, sum-, and function-type.

$$\begin{aligned}
\mathcal{V}[\![\tau_1 \times \tau_2]\!]_\rho(v_1, v_2) &\triangleq (v_1, v_2) \in \{((w_{11}, w_{12}), (w_{21}, w_{22})) \mid \mathcal{V}[\![\tau_1]\!]_\rho(w_{11}, w_{21}) \wedge \\
&\qquad\qquad \mathcal{V}[\![\tau_2]\!]_\rho(w_{12}, w_{22})\} \\
\mathcal{V}[\![\tau_1 + \tau_2]\!]_\rho(v_1, v_2) &\triangleq (v_1, v_2) \in \{(\mathsf{inj}_1\ w_1, \mathsf{inj}_1\ w_2) \mid \mathcal{V}[\![\tau_1]\!]_\rho(w_1, w_2)\} \vee \\
&\qquad (v_1, v_2) \in \{(\mathsf{inj}_2\ w_1, \mathsf{inj}_2\ w_2) \mid \mathcal{V}[\![\tau_2]\!]_\rho(w_1, w_2)\} \\
\mathcal{V}[\![\tau_1 \to \tau_2]\!]_\rho(v_1, v_2) &\triangleq (v_1, v_2) \in \{(\lambda x.\ e_1, \lambda x.\ e_2) \mid \forall w_1, w_2. \mathcal{V}[\![\tau_1]\!]_\rho(w_1, w_2) \implies \\
&\qquad\qquad \mathcal{E}[\![\tau_2]\!]_\rho(e_1[w_1/x], e_2[w_2/x])\}
\end{aligned}$$

Note how the design principle of the property being preserved by eliminating forms comes into play for these three types – for product type, the two pairs are

related if through both eliminating forms, the resulting values are related.
In the relation for function types, we mention another relation $\mathcal{E}$. This is the expression interpretation which we will define in a bit. Informally, two expressions are in our expression interpretation when they terminate at related values.
Finally, let's discuss the perhaps most intricate type: type abstraction.

$$\mathcal{V}[\![\forall X.\ \tau]\!]_\rho(v_1, v_2) \triangleq (v_1, v_2) \in \{(\Lambda\ e_1, \Lambda\ e_2) \mid \forall \tau_1, \tau_2, R \in Rel[\tau_1, \tau_2]. \\ \mathcal{E}[\![\tau]\!]_{\rho[X \mapsto (\tau_1, \tau_2, R)]}(e_1, e_2)\}$$

For type abstraction, the eliminating form simply removes the type abstraction, so nothing interesting happens with regards to the expressions. However, we must also state the type that the two inner expressions are related at. Looking at typing rule T-Tapp, eliminating a type abstraction means we must substitute the type variable with some type. So what should we substitute X for in τ? We do not want to limit ourselves to saying that both expressions must use the *same* type when eliminating the type-abstraction. Doing so is a valid choice, but it makes our logical relation less powerful. Instead, we shall allow the expressions to be eliminated using *any* two types τ_1, τ_2. But we still can't use either τ_1 or τ_2 to substitute for the type variable – this would break well-typedness for one of the two expressions. The solution is to not perform the substitution at all and let the type they are related under, τ be open. This has the implication that we could now be interpreting values at open types. So we must of course close off the types when we have to use them in the well-typed relation. To remember our choice of types τ_1, τ_2, we introduce the relational type substitution $\rho = \{X \mapsto (\tau_1, \tau_2, R_1), Y \mapsto (\tau_3, \tau_4, R_2), \dots\}$. The first two projections of this, $\rho_1 = \{X \mapsto \tau_1, Y \mapsto \tau_3, \dots\}$ and $\rho_2 = \{X \mapsto \tau_2, Y \mapsto \tau_4, \dots\}$ then become type substitutions. So if we have multiple type abstractions in our two expressions, then ρ_1 closes the type on the left with the types we have decided previously, and similarly for ρ_2, but for the type on the right.
This solves the problem of what type our expressions should be related at, but it introduces another problem – since we have open types, then we need an interpretation for pairs of values under a type variable, X. But how do we decide which values are related under the type variable? The solution is to decide those values in the interpretation for the type abstraction. We shall in there pick a relation R, which tells us which pairs of values we consider to be related under the type variable. R can be any relation in $Rel[\tau_1, \tau_2]$, which is defined as follows.

$$Rel[\tau_1, \tau_2] \triangleq \left\{ R \;\middle|\; \begin{array}{l} R \in \mathcal{P}(Val \times Val) \land \bullet \mid \bullet \vdash \tau_1 \land \bullet \mid \bullet \vdash \tau_2 \land \\ \forall (v_1, v_2) \in R.\ \bullet \mid \bullet \vdash v_1 : \tau_1 \land \bullet \mid \bullet \vdash v_2 : \tau_2 \end{array} \right\}$$

This set is not very restrictive. So in proofs, if have two type abstractions that we know are logically related, then we get many options for choosing what R should be, and choosing the right R will be a deciding factor in whether or not the proof goes through.
Now, since we have decided which values are related, then, when encountering a type variable, say X, we can look up the relation in the third projection of ρ, $\rho_R = \{X \mapsto R_1, Y \mapsto R_2, \dots\}$, and then we simply state that a pair of values

are in our logical relation, if they are in the relation R_1 we chose. Thus, the value interpretation for type variables is as follows.

$$\mathcal{V}[\![X]\!]_\rho(v_1, v_2) \triangleq (v_1, v_2) \in \rho_R[X]$$

Now we are ready to define the expression interpretation.

$$\begin{aligned}\mathcal{E}[\![\tau]\!]_\rho(e_1, e_2) \triangleq \bullet \mid \bullet \vdash e_1 : \rho_1(\tau) \wedge \bullet \mid \bullet \vdash e_2 : \rho_2(\tau) \wedge \\ (\exists v_1, v_2 . e_1 \to^* v_1 \wedge e_2 \to^* v_2 \wedge \mathcal{V}[\![\tau]\!]_\rho(v_1, v_2))\end{aligned}$$

This states that two closed expressions are logically related when they are well-typed and reduce to logically related values. This makes intuitive sense, as we have defined our value interpretation so that all pairs of values are contextually equivalent, and if two expressions run to contextually equivalent values, then we consider those two expressions to be contextually equivalent as well.

Next, let's define the interpretations for type environments Ξ and variable environments Γ. These will be a little different in that they will be sets of relational substitutions, which allow us to close off open types and open expressions. First, we define the interpretation for type environments inductively as follows.

$$\begin{aligned}\mathcal{D}[\![\bullet]\!] &\triangleq \{\emptyset\} \\ \mathcal{D}[\![\Xi, X]\!] &\triangleq \{\rho[X \mapsto (\tau_1, \tau_2, R)] \mid \rho \in \mathcal{D}[\![\Xi]\!] \wedge R \in Rel[\tau_1, \tau_2]\}\end{aligned}$$

Secondly, our interpretation for variable environments is as follows.

$$\begin{aligned}\mathcal{G}[\![\bullet]\!]_\rho &\triangleq \{\emptyset\} \\ \mathcal{G}[\![\Gamma, x : \tau]\!]_\rho &\triangleq \{\gamma[x \mapsto (v_1, v_2)] \mid \gamma \in \mathcal{G}[\![\Gamma]\!]_\rho \wedge \mathcal{V}[\![\tau]\!]_\rho(v_1, v_2)\}\end{aligned}$$

Like with the relational type substitution, we also have the two projections of γ, $\gamma_1 = \{x \mapsto v_1, \dots\}$ and $\gamma_2 = \{x \mapsto v_2, \dots\}$. Note that this definition ensures that we close off two expressions only with related values.
With these interpretations defined, we are now finally ready to give our complete definition of our LR model.

Definition 4.2.1 (Logical Relations Model)**.**

$$\begin{gathered}\Xi \mid \Gamma \vdash e \approx^{LR} e' : \tau \\ \iff \\ \Xi \mid \Gamma \vdash e : \tau \;\wedge\; \Xi \mid \Gamma \vdash e' : \tau \wedge \\ \forall \rho \in \mathcal{D}[\![\Xi]\!], \gamma \in \mathcal{G}[\![\Gamma]\!]_\rho . \mathcal{E}[\![\tau]\!]_\rho(\gamma_1(e), \gamma_2(e'))\end{gathered}$$

Thus, two expressions e and e' are logically related at type τ under Ξ and Γ, when both expressions are well typed at type τ under Ξ and Γ, and when we can choose *any* relational type substitution $\rho \in \mathcal{D}[\![\Xi]\!]$ and *any* relational variable substitution $\gamma \in \mathcal{G}[\![\Gamma]\!]_\rho$, and the expressions closed off using γ are in our expression interpretation at type τ with our relational type substitution ρ.
For the remainder, we shall refer to the relation simply as "LR".

4.3 Compatibility Lemmas

As mentioned in the first section of this chapter, we want to show that LR is a congruency relation. So we must go through each congruency rule mentioned in appendix A (but with the relation being LR) and show that it holds. We refer to these lemmas as "compatibility lemmas". Some compatibility lemmas are very similar, so we show only a selected few. Two additional compatibility lemmas can be found in appendix C. For the sake of brevity, we only show well-typedness in the first two compatibility lemmas.

Lemma 12 (Cmpt-unit). $\dfrac{}{\Xi \mid \Gamma \vdash () \approx^{LR} () : \mathsf{Unit}}$

Proof. By definition 4.2.1, we must show $\Xi \mid \Gamma \vdash () : \mathsf{Unit} \land \Xi \mid \Gamma \vdash () : \mathsf{Unit} \land \forall \rho \in \mathcal{D}[\![\Xi]\!], \gamma \in \mathcal{G}[\![\Gamma]\!]_\rho.\mathcal{E}[\![\mathsf{Unit}]\!]_\rho(\gamma_1(()), \gamma_2(()))$. The well-typedness simply follows by the typing rule T-unit. So assume some $\rho \in \mathcal{D}[\![\Xi]\!]$ and $\gamma \in \mathcal{G}[\![\Gamma]\!]_\rho$, we must show $\mathcal{E}[\![\mathsf{Unit}]\!]_\rho(\gamma_1(()), \gamma_2(())) \equiv \mathcal{E}[\![\mathsf{Unit}]\!]_\rho((), ())$. Thus, we must show three things

1. $\bullet \mid \bullet \vdash () : \rho_1(\mathsf{Unit})$
2. $\bullet \mid \bullet \vdash () : \rho_2(\mathsf{Unit})$
3. $\exists v_1, v_2.() \rightarrow^* v_1 \land () \rightarrow^* v_2 \land \mathcal{V}[\![\mathsf{Unit}]\!]_\rho(v_1, v_2)$

Given that $\rho_1(\mathsf{Unit}) = \rho_2(\mathsf{Unit}) = \mathsf{Unit}$, the well-typedness once again simply follows from typing rule T-unit. For (3), we choose $()$ for v_1 and v_2. Then we must show $() \rightarrow^* () \land () \rightarrow^* () \land \mathcal{V}[\![\mathsf{Unit}]\!]_\rho((), ())$. The evaluations hold trivially. The last part says we must show $\mathcal{W}[\![\mathsf{Unit}]\!]_\rho((), ()) \land ((), ()) \in \{((), ())\}$ (remember that $\mathcal{W}[\![\mathsf{Unit}]\!]_\rho((), ())$ was implicit). The last part trivially holds, and the first part holds by the same argument made above: $\rho_1(\mathsf{Unit}) = \rho_2(\mathsf{Unit}) = \mathsf{Unit}$, and then it follows from T-unit. $\square$

Lemma 13 (Cmpt-add). $\dfrac{\Xi \mid \Gamma \vdash e_1 \approx^{LR} e_1' : \mathbb{Z} \qquad \Xi \mid \Gamma \vdash e_2 \approx^{LR} e_2' : \mathbb{Z}}{\Xi \mid \Gamma \vdash e_1 + e_2 \approx^{LR} e_1' + e_2' : \mathbb{Z}}$

Proof. So assuming

$$\Xi \mid \Gamma \vdash e_1 \approx^{LR} e_1' : \mathbb{Z} \tag{4.1}$$

$$\Xi \mid \Gamma \vdash e_2 \approx^{LR} e_2' : \mathbb{Z} \tag{4.2}$$

we must show $\Xi \mid \Gamma \vdash e_1 + e_2 : \mathbb{Z} \land \Xi \mid \Gamma \vdash e_1' + e_2' : \mathbb{Z} \land \forall \rho \in \mathcal{D}[\![\Xi]\!], \gamma \in \mathcal{G}[\![\Gamma]\!]_\rho.\mathcal{E}[\![\mathbb{Z}]\!]_\rho(\gamma_1(e_1 + e_2), \gamma_2(e_1' + e_2'))$. From (4.1), we have $\Xi \mid \Gamma \vdash e_1 : \mathbb{Z}$, and from (4.2) we have $\Xi \mid \Gamma \vdash e_2 : \mathbb{Z}$. So by the T-add rule, we get $\Xi \mid \Gamma \vdash e_1 + e_2 : \mathbb{Z}$. The well-typedness of $e_1' + e_2'$ follows in the same way.
So let's assume some $\rho \in \mathcal{D}[\![\Xi]\!]$ and $\gamma \in \mathcal{G}[\![\Gamma]\!]_\rho$. We must show $\mathcal{E}[\![\mathbb{Z}]\!]_\rho(\gamma_1(e_1 + e_2), \gamma_2(e_1' + e_2'))$, which is equivalent to $\mathcal{E}[\![\mathbb{Z}]\!]_\rho(\gamma_1(e_1) + \gamma_1(e_2), \gamma_2(e_1') + \gamma_2(e_2'))$. Unfolding the expression interpretation, what we need to show is the following three points.

1. $\bullet \mid \bullet \vdash \gamma_1(e_1) + \gamma_1(e_2) : \rho_1(\mathbb{Z})$
2. $\bullet \mid \bullet \vdash \gamma_2(e_1') + \gamma_2(e_2') : \rho_2(\mathbb{Z})$
3. $\exists v_1, v_2.\gamma_1(e_1) + \gamma_1(e_2) \to^* v_1 \land \gamma_2(e_1') + \gamma_2(e_2') \to^* v_2 \land \mathcal{V}[\![\mathbb{Z}]\!]_\rho(v_1, v_2)$

To prove these points, we will need some more information. From the last part of (4.1) instantiated with ρ and γ, we get

$$\bullet \mid \bullet \vdash \gamma_1(e_1) : \rho_1(\mathbb{Z}) \tag{4.3}$$
$$\bullet \mid \bullet \vdash \gamma_2(e_1') : \rho_2(\mathbb{Z}) \tag{4.4}$$
$$\exists v_1, v_2.\gamma_1(e_1) \to^* v_1 \land \gamma_2(e_1') \to^* v_2 \land \mathcal{V}[\![\mathbb{Z}]\!]_\rho(v_1, v_2) \tag{4.5}$$

Likewise, from the last part of 4.2 instantiated with ρ and γ, we get

$$\bullet \mid \bullet \vdash \gamma_1(e_2) : \rho_1(\mathbb{Z}) \tag{4.6}$$
$$\bullet \mid \bullet \vdash \gamma_2(e_2') : \rho_2(\mathbb{Z}) \tag{4.7}$$
$$\exists v_1, v_2.\gamma_1(e_2) \to^* v_1 \land \gamma_2(e_2') \to^* v_2 \land \mathcal{V}[\![\mathbb{Z}]\!]_\rho(v_1, v_2) \tag{4.8}$$

Now to prove the three points above. Given that $\mathbb{Z}$ is a closed type, $\rho_1(\mathbb{Z}) = \rho_2(\mathbb{Z}) = \mathbb{Z}$. Thus, point 1 follows from (4.3), (4.6), and the T-add rule. Similarly, point 2 follows from (4.4), (4.7), and the T-add rule.
So all that remains to be shown is point 3. Let's denote the values in (4.5) v_1' and v_2', and the values in (4.8) v_1'' and v_2''. Then we know

$$\gamma_1(e_1) + \gamma_1(e_2) \to^* v_1' + \gamma_1(e_2) \to^* v_1' + v_1'' \tag{4.9}$$
$$\gamma_2(e_1') + \gamma_2(e_2') \to^* v_2' + \gamma_2(e_2') \to^* v_2' + v_2'' \tag{4.10}$$
$$\mathcal{V}[\![\mathbb{Z}]\!]_\rho(v_1', v_2') \land \mathcal{V}[\![\mathbb{Z}]\!]_\rho(v_1'', v_2'') \tag{4.11}$$

From (4.11), we know that all our values are integers and $\overline{v_1'} = \overline{v_2'}$, and $\overline{v_1''} = \overline{v_2''}$. Furthermore $\overline{v_1'} + \overline{v_1''} \to \overline{v_1' + v_1''}$, and $\overline{v_2'} + \overline{v_2''} \to \overline{v_2' + v_2''}$. Thus $\gamma_1(e_1) + \gamma_1(e_2) \to^* \overline{v_1' + v_1''}$ and $\gamma_2(e_1') + \gamma_2(e_2') \to^* \overline{v_2' + v_2''}$.
What we need to show is point 3. So now use $\overline{v_1' + v_1''}$ for v_1, and $\overline{v_2' + v_2''}$ for v_2. The first two propositions hold by what we have just shown, so all that remains is to show $\mathcal{V}[\![\mathbb{Z}]\!]_\rho(\overline{v_1' + v_1''}, \overline{v_2' + v_2''})$, which by definition means we have to show $\mathcal{W}[\![\mathbb{Z}]\!]_\rho(\overline{v_1' + v_1''}, \overline{v_2' + v_2''})$ and $v_1' + v_1'' = v_2' + v_2''$. The first part follows simply from the T-int rule. For the second part, note that we know $\overline{v_1'} = \overline{v_2'}$, and $\overline{v_1''} = \overline{v_2''}$, so it follows that $\overline{v_1' + v_1''} = \overline{v_2' + v_1''} = \overline{v_2' + v_2''}$. ☐

Lemma 14 (Cmpt-fst). $\dfrac{\Xi \mid \Gamma \vdash e \approx^{LR} e' : \tau_1 \times \tau_2}{\Xi \mid \Gamma \vdash \mathsf{fst}\ e \approx^{LR} \mathsf{fst}\ e' : \tau_1}$

Proof. So assuming $\Xi \mid \Gamma \vdash e \approx^{LR} e' : \tau_1 + \tau_2$, we must show (ignoring well-typedness): $\forall \rho \in \mathcal{D}[\![\Xi]\!], \gamma \in \mathcal{G}[\![\Gamma]\!]_\rho.\mathcal{E}[\![\tau_1]\!]_\rho(\gamma_1(\mathsf{fst}\ e), \gamma_2(\mathsf{fst}\ e'))$. So assume some $\rho \in \mathcal{D}[\![\Xi]\!]$, and $\gamma \in \mathcal{G}[\![\Gamma]\!]_\rho$, we must show $\mathcal{E}[\![\tau_1]\!]_\rho(\gamma_1(\mathsf{fst}\ e), \gamma_2(\mathsf{fst}\ e')) \equiv \mathcal{E}[\![\tau_1]\!]_\rho(\mathsf{fst}\ \gamma_1(e), \mathsf{fst}\ \gamma_2(e'))$. By definition of our expression interpretation, this corresponds to showing

$$\exists v_1, v_2.\mathsf{fst}\ \gamma_1(e) \to^* v_1 \wedge \mathsf{fst}\ \gamma_2(e') \to^* v_2 \wedge \mathcal{V}[\![\tau_1]\!]_\rho(v_1, v_2) \tag{4.12}$$

Note first that if we instantiate our initial assumption with ρ and γ, we get

$$\exists v_1, v_2.\gamma_1(e) \to^* v_1 \wedge \gamma_2(e') \to^* v_2 \wedge \mathcal{V}[\![\tau_1 \times \tau_2]\!]_\rho(v_1, v_2) \tag{4.13}$$

Let's instantiate the values in (4.13) as v_1' and v_2'. Then we know

$$\mathsf{fst}\ \gamma_1(e) \to^* \mathsf{fst}\ v_1' \tag{4.14}$$

$$\mathsf{fst}\ \gamma_2(e') \to^* \mathsf{fst}\ v_2' \tag{4.15}$$

$$\mathcal{V}[\![\tau_1 \times \tau_2]\!]_\rho(v_1', v_2') \tag{4.16}$$

By (4.16) we know that $v_1' = (w_{11}, w_{12})$ and $v_2' = (w_{21}, w_{22})$ for some values $w_{11}, w_{12}, w_{21}, w_{22}$. Further, $\mathcal{V}[\![\tau_1]\!]_\rho(w_{11}, w_{21})$ and $\mathcal{V}[\![\tau_2]\!]_\rho(w_{12}, w_{22})$. Now, by (4.14) and E-fst we have $\mathsf{fst}\ \gamma_1(e) \to^* w_{11}$. Likewise, from (4.15) and E-fst, we get $\mathsf{fst}\ \gamma_2(e') \to^* w_{21}$.
Now to prove the goal (4.12). We take w_{11} for v_1 and w_{21} for v_2, so that we must show $\mathsf{fst}\ \gamma_1(e) \to^* w_{11} \wedge \mathsf{fst}\ \gamma_2(e') \to^* w_{21} \wedge \mathcal{V}[\![\tau_1]\!]_\rho(w_{11}, w_{21})$, all of which we have just argued holds. $\square$

Lemma 15 (Cmpt-lam). $\dfrac{\Xi \mid \Gamma, x : \tau_1 \ \vdash e \approx^{LR} e' : \tau_2}{\Xi \mid \Gamma \ \vdash \lambda x.\ e \approx^{LR} \lambda x.\ e' : \tau_1 \to \tau_2}$

Proof. So assuming

$$\Xi \mid \Gamma, x : \tau_1 \ \vdash e \approx^{LR} e' : \tau_2 \tag{4.17}$$

we must show: $\forall \rho \in \mathcal{D}[\![\Xi]\!], \gamma \in \mathcal{G}[\![\Gamma]\!]_\rho.\mathcal{E}[\![\tau_1 \to \tau_2]\!]_\rho(\gamma_1(\lambda x.\ e), \gamma_2(\lambda x.\ e'))$. So assume some $\rho \in \mathcal{D}[\![\Xi]\!]$, and $\gamma \in \mathcal{G}[\![\Gamma]\!]_\rho$, then we must show $\mathcal{E}[\![\tau_1 \to \tau_2]\!]_\rho(\gamma_1(\lambda x.\ e), \gamma_2(\lambda x.\ e')) \equiv \mathcal{E}[\![\tau_1 \to \tau_2]\!]_\rho(\lambda x.\ \gamma_1(e), \lambda x.\ \gamma_2(e'))$, which by definition corresponds to

$$\exists v_1, v_2.\lambda x.\ \gamma_1(e) \to^* v_1 \wedge \lambda x.\ \gamma_2(e') \to^* v_2 \wedge \mathcal{V}[\![\tau_1 \to \tau_2]\!]_\rho(v_1, v_2) \tag{4.18}$$

Since $\lambda x.\ \gamma_1(e)$ and $\lambda x.\ \gamma_2(e')$ are already values, we can use these for v_1 and v_2. The first two parts are then trivially satisfied, so we only need to show $\mathcal{V}[\![\tau_1 \to \tau_2]\!]_\rho(\lambda x.\ \gamma_1(e), \lambda x.\ \gamma_2(e'))$, which by definition means we have to show $\forall w_1, w_2.\mathcal{V}[\![\tau_1]\!]_\rho(w_1, w_2) \implies \mathcal{E}[\![\tau_2]\!]_\rho(\gamma_1(e)[w_1/x], \gamma_2(e')[w_2/x])$. So assume some w_1, w_2 such that $\mathcal{V}[\![\tau_1]\!]_\rho(w_1, w_2)$, then we must show $\mathcal{E}[\![\tau_2]\!]_\rho(\gamma_1(e)[w_1/x], \gamma_2(e')[w_2/x])$, which by definition means we have to show $\exists v_1, v_2.\gamma_1(e)[w_1/x] \to^* v_1 \wedge \gamma_2(e')[w_2/x] \to^* v_2 \wedge \mathcal{V}[\![\tau_2]\!]_\rho(v_1, v_2)$. Now we instantiate (4.17) with $\rho, \gamma' = \gamma[x \mapsto (w_1, w_2)]$, so that we have $\exists v_1, v_2.\gamma_1'(e) \to^* v_1 \wedge \gamma_2'(e') \to^* v_2 \wedge \mathcal{V}[\![\tau_2]\!]_\rho(v_1, v_2)$. By the substitution lemma (lemma 3), this is equivalent to our goal, so we are done. $\square$

Lemma 16 (Cmpt-Tlam). $\dfrac{\Xi, X \mid \Gamma \vdash e \approx^{LR} e' : \tau}{\Xi \mid \Gamma \vdash \Lambda\, e \approx^{LR} \Lambda\, e' : \forall X.\, \tau}$

Proof. Assuming

$$\Xi, X \mid \Gamma \vdash e \approx^{LR} e' : \tau \tag{4.19}$$

we must show $\forall \rho \in \mathcal{D}[\![\Xi]\!], \gamma \in \mathcal{G}[\![\Gamma]\!]_\rho . \mathcal{E}[\![\forall X.\, \tau]\!]_\rho(\gamma_1(\Lambda\ e), \gamma_2(\Lambda\ e'))$. So assume some $\rho \in \mathcal{D}[\![\Xi]\!]$ and $\gamma \in \mathcal{G}[\![\Gamma]\!]_\rho$, by the definition of expression interpretation, our goal is now to show

$$\exists v_1, v_2 . \Lambda\ \gamma_1(e) \to^* v_1 \wedge \Lambda\ \gamma_2(e') \to^* v_2 \wedge \mathcal{V}[\![\forall X.\, \tau]\!]_\rho(v_1, v_2) \tag{4.20}$$

Note here that we have pushed the variable substitution inside the Λ-abstraction. Here we can simply use $\Lambda\ \gamma_1(e)$ for v_1 and $\Lambda\ \gamma_2(e')$ for v_2, as they are already values. Thus, we must show $\mathcal{V}[\![\forall X.\, \tau]\!]_\rho(\Lambda\ \gamma_1(e), \Lambda\ \gamma_2(e'))$, which by definition means we have to show $\forall \tau_1, \tau_2, R \in Rel[\tau_1, \tau_2] . \mathcal{E}[\![\tau]\!]_{\rho'}(\gamma_1(e), \gamma_2(e'))$, where $\rho' = \rho[X \mapsto (\tau_1, \tau_2, R)]$. So assume some types τ_1, τ_2, and a relation $R \in Rel[\tau_1, \tau_2]$, then our goal is

$$\mathcal{E}[\![\tau]\!]_{\rho'}(\gamma_1(e), \gamma_2(e')) \tag{4.21}$$

We now instantiate (4.19) with ρ' and γ. Note that $\gamma \in \mathcal{G}[\![\Gamma]\!]_{\rho'}$ by corollary 28.1. Thus, we get $\mathcal{E}[\![\tau]\!]_{\rho'}(\gamma_1(e), \gamma_2(e'))$, which is exactly what we had to show. □

Lemma 17 (Cmpt-Tapp). $\dfrac{\Xi \mid \Gamma \vdash e \approx^{LR} e' : \forall X.\, \tau}{\Xi \mid \Gamma \vdash e\ \_ \approx^{LR} e'\ \_ : \tau[\tau'/X]}$

Proof. Similar to proof given in [2]. □

4.4 Properties of LR

With the compatibility lemmas proved, we get that LR is a congruency relation

Lemma 18 (Congruency of LR). *LR is a congruence relation*

Next, we shall show that LR is an adequate relation.

Lemma 19 (Adequacy of LR). $\bullet \mid \bullet \vdash e \approx^{LR} e' : \mathbb{B} \implies e \Downarrow v \iff e' \Downarrow v$

Proof. Assuming $\bullet \mid \bullet \vdash e \approx^{LR} e' : \mathbb{B}$, we must show $e \Downarrow v \iff e' \Downarrow v$. By the definition of our LR model, we have that $\forall \rho \in \mathcal{D}[\![\bullet]\!], \gamma \in \mathcal{G}[\![\bullet]\!]_\rho . \mathcal{E}[\![\mathbb{B}]\!]_\rho(\gamma_1(e), \gamma_2(e'))$. Given our definitions of $\mathcal{D}$ and $\mathcal{G}$, we can only instantiate it with the empty relational type substitution and the empty relational variable substitution. Doing this, we get $\mathcal{E}[\![\mathbb{B}]\!]_\emptyset(\emptyset(e), \emptyset(e')) \equiv \mathcal{E}[\![\mathbb{B}]\!]_\emptyset(e, e')$, which means we have $\exists v_1, v_2 . e \to^* v_1 \wedge e' \to^* v_2 \wedge \mathcal{V}[\![\mathbb{B}]\!]_\emptyset(v_1, v_2)$. Instantiating these values, we may conclude that $e \Downarrow v_1$, $e' \Downarrow v_2$. Now, our goal is to show $e \Downarrow v \iff e' \Downarrow v$. Both sides are similar, so we show just the " $\implies$ " direction. So we assume $e \Downarrow v$, and must show $e' \Downarrow v$. By corollary 6.1, we may conclude that $v_1 = v$, so that our goal now is to show $e' \Downarrow v_1$. Given we know that $e' \Downarrow v_2$, then it suffices to show that $v_1 = v_2$. Since $\mathcal{V}[\![\mathbb{B}]\!]_\emptyset(v_1, v_2)$, then either $v_1 = v_2 = \textsf{true}$ or $v_1 = v_2 = \textsf{false}$. In any case, $v_1 = v_2$, so we are done. □

We are now finally ready prove that our LR model is sound wrt. contextual equivalence.

Theorem 20. $\Xi \mid \Gamma \vdash e \approx^{LR} e' : \tau \implies \Xi \mid \Gamma \vdash e \approx^{ctx} e' : \tau$

Proof. We now know that LR is an adequate relation (lemma 19) and a congruence relation (lemma 18). Further, it follows directly from the definition of LR, that all expressions in the relation are well-typed. Thus, by theorem 11, we may conclude $\Xi \mid \Gamma \vdash e \approx^{LR} e' : \tau \implies \Xi \mid \Gamma \vdash e \approx^{ctx} e' : \tau$. □

We may also show the following theorem which is usually referred to as the fundamental theorem.

Theorem 21 (Fundamental Theorem)**.** $\Xi \mid \Gamma \vdash e : \tau \implies \Xi \mid \Gamma \vdash e \approx^{LR} e : \tau$

Proof. Since LR is a congruence relation (lemma 18), it follows directly from lemma 9. □

The fundamental theorem may not seem it, but it is actually quite powerful. We shall see this in great detail in the next chapter, but to illustrate the point, let's now prove the normalisation theorem from chapter 2.

proof of theorem 5. Given a closed and well-typed expression, $\bullet \mid \bullet \vdash e : \tau$, we wish to show that e reduces to normal form. By the fundamental theorem, we know that $\bullet \mid \bullet \vdash e \approx^{LR} e : \tau$, which by definition gives us $\forall \rho \in \mathcal{D}[\![\bullet]\!], \gamma \in \mathcal{G}[\![\bullet]\!]_\rho . \mathcal{E}[\![\tau]\!]_\rho(\gamma_1(e), \gamma_2(e))$. We only have one choice of instantiation; the empty relational substitutions. Thus, we get $\mathcal{E}[\![\tau]\!]_\emptyset(\emptyset(e), \emptyset(e)) \equiv \mathcal{E}[\![\tau]\!]_\emptyset(e, e)$. By the definition of our expression interpretation, this means that $e \rightarrow^* v$, for some v. Thus e reduces to normal form. □

Chapter 5

Examples of Application of Contextual Equivalence

With our logical relations model defined and the necessary properties of it proved, we are ready to put it to use on some interesting examples. Some of the theorems will involve contextual equivalence directly, but as we will see, we can even use it in theorems where it isn't mentioned! This is thanks to the fundamental theorem.

5.1 Identity

The first example we shall explore is that of identity; if an expression has a certain type, then we may prove that it behaves just as the identity function. We shall actually show two theorems. The first one is weaker, in the sense that there are some languages (fx ones with side-effects), where only the first theorem holds.

5.1.1 Identity: Reduction

Informally, the first theorem says that an expression e, which given a value of any type τ outputs an expression of the same type τ, will always output the value it was given as input. The intuition for why this is, is that since the function works for any type, it cannot do anything specific to the value given. Since it must output an expression of the same type, and it doesn't know in advance the type of the input, it only has one option: to return the value it was given. A theorem like this, where we can conclude something about an expression just by inspecting it's type, is called a *free theorem*[3], and a standard way to prove these is using logical relations.

Theorem 22. $\bullet \mid \bullet \vdash e : \forall X.\, X \to X \wedge \bullet \mid \bullet \vdash v : \tau \implies (e\ \_)\ v \to^* v$

Proof. So assume

$$\bullet \mid \bullet \vdash e : \forall X.\ X \to X \tag{5.1}$$

$$\bullet \mid \bullet \vdash v : \tau \tag{5.2}$$

Then we must show $(e\ \_)\ v \to^* v$. From 5.1 and by the fundamental theorem (theorem 21) we know $\bullet \mid \bullet \ \vdash e \approx^{LR} e : \forall X.\ X \to X$ from which we get $\forall \rho \in \mathcal{D}[\![\bullet]\!], \gamma \in \mathcal{G}[\![\bullet]\!]_\rho.\mathcal{E}[\![\forall X.\ X \to X]\!]_\rho(\gamma_1(e), \gamma_2(e))$. We instantiate this with our only choice, the empty relational substitutions, $\emptyset$. Thus we get $\mathcal{E}[\![\forall X.\ X \to X]\!]_\emptyset(e, e)$. And from this, we may conclude $\exists v_1, v_2.e \to^* v_1 \wedge e \to^* v_2 \wedge \mathcal{V}[\![\forall X.\ X \to X]\!]_\emptyset(v_1, v_2)$. By determinacy (specifically, corollary 6.1) these values must be the same, and from the value interpretation we further know that the value must be a type abstraction. Thus $e \to^* \Lambda\ e'$ and $\forall \tau_1, \tau_2, R \in \mathit{Rel}[\tau_1, \tau_2].\mathcal{E}[\![X \to X]\!]_{\rho[X \mapsto (\tau_1, \tau_2, R)]}(e', e')$. Here we use τ for τ_1 and τ_2, and the relation $R' = \{(v, v)\}$ for R, giving us $\mathcal{E}[\![X \to X]\!]_{\rho[X \mapsto (\tau_1, \tau_2, R')]}(e', e')$, meaning $\exists v'_1, v'_2.e' \to^* v'_1 \wedge e' \to^* v'_2 \wedge \mathcal{V}[\![X \to X]\!]_{\rho[X \mapsto (\tau_1, \tau_2, R')]}(v'_1, v'_2)$. Thus $e' \to^* \lambda x.\ e''$ and $\forall w_1, w_2.\mathcal{V}[\![X]\!]_{\rho[X \mapsto (\tau_1, \tau_2, R')]}(w_1, w_2) \implies \mathcal{E}[\![X]\!]_{\rho[X \mapsto (\tau_1, \tau_2, R')]}(e''[w_1/x], e''[w_2/x])$. Using v for both w_1 and w_2, we get $\mathcal{E}[\![X]\!]_{\rho[X \mapsto (\tau_1, \tau_2, R')]}(e''[v/x], e''[v/x])$ which means $\exists v_{1f}, v_{2f}.e''[v/x] \to^* v_{1f} \wedge e''[v/x] \to^* v_{2f} \wedge \mathcal{V}[\![X]\!]_{\rho[X \mapsto (\tau_1, \tau_2, R')]}(v_{1f}, v_{2f})$, which again means $e''[v/x] \to^* v_f$ and $(v_f, v_f) \in R'$, but since R' is a singleton, then we must have $v_f = v$. From the above, we have the following evaluation

$$(e\ \_)\ v \to^* ((\Lambda\ e')\ \_)\ v \to e'\ v \to^* (\lambda x.\ e'')\ v \to e''[v/x] \to^* v_f = v \tag{5.3}$$

The first "$\to^*$" follows from $e \to^* \Lambda\ e'$. The following step holds by E-tapp-tlam. The next "$\to^*$" follows by $e' \to^* \lambda x.\ e''$. The next step holds by E-lam-app. and the final "$\to^*$" follows by $e''[v/x] \to^* v_f$. Thus we have know shown $(e\ \_)\ v \to^* v$. □

5.1.2 Identity: Contextual Equivalence

The next theorem shows that any expression of type $\forall X.\ X \to X$ may as well be substituted with the simple, generic identity function.

Theorem 23. $\Xi \mid \Gamma \ \vdash e : \forall X.\ X \to X \implies \Xi \mid \Gamma \ \vdash e \approx^{ctx} \Lambda\ \lambda x.\ x : \forall X.\ X \to X$

Proof. The majority of this proof is simply unfolding definitions and applying the fundamental theorem. So we shall go fairly quickly through this.
So assuming $\Xi \mid \Gamma \ \vdash e : \forall X.\ X \to X$, we must show $\Xi \mid \Gamma \ \vdash e \approx^{ctx} \Lambda\ \lambda x.\ x : \forall X.\ X \to X$. We shall do this by showing that the two expressions are logically related. Then contextual equivalence follows from theorem 20. So we assume some $\rho \in \mathcal{D}[\![\Xi]\!]$ and $\gamma \in \mathcal{G}[\![\Gamma]\!]_\rho$. Then we must show $\exists v_1, v_2.\gamma_1(e) \to^* v_1 \wedge \Lambda\ \lambda x.\ x \to^* v_2 \wedge \mathcal{V}[\![\forall X.\ X \to X]\!]_\rho(v_1, v_2)$. By the fundamental theorem on

our initial assumption and by instantiating this with ρ and γ, we may conclude $\bullet \mid \bullet \vdash \gamma_2(e) : \forall X.\ X \to X$ and $\gamma_1(e) \to^* \Lambda\ e_1$ and $\gamma_2(e) \to^* \Lambda\ e_2$ and

$$\forall \tau_1, \tau_2, R \in Rel[\tau_1, \tau_2].\mathcal{E}[\![X \to X]\!]_{\rho[X \mapsto (\tau_1, \tau_2, R)]}(e_1, e_2) \tag{5.4}$$

Now use $\Lambda\ e_1$ for v_1 and $\Lambda\ \lambda x.\ x$ for v_2 in our goal. Then we must show $\mathcal{V}[\![\forall X.\ X \to X]\!]_\rho(\Lambda\ e_1, \Lambda\ \lambda x.\ x)$. So assume some τ_1, τ_2, and $R \in Rel[\tau_1, \tau_2]$, then we must show $\exists v_1', v_2'.e_1 \to^* v_1' \wedge \lambda x.\ x \to^* v_2' \wedge \mathcal{V}[\![X \to X]\!]_{\rho'}(v_1', v_2')$, where $\rho' = \rho[X \mapsto (\tau_1, \tau_2, R)]$. Now instantiate (5.4) with τ_1, τ_2, R. Then we know $e_1 \to^* \lambda x.\ e_1'$, and $e_2 \to^* \lambda x.\ e_2'$, and

$$\forall w_1, w_2.\mathcal{V}[\![X]\!]_{\rho'}(w_1, w_2) \implies \mathcal{E}[\![X]\!]_{\rho'}(e_1'[w_1/x], e_2'[w_2/x]) \tag{5.5}$$

We then use $\lambda x.\ e_1'$ for v_1' and $\lambda x.\ x$ for v_2' in our goal. Then we have to show $\mathcal{V}[\![X \to X]\!]_{\rho'}(\lambda x.\ e_1', \lambda x.\ x)$. So assume some w_1 and w_2, such that $\mathcal{V}[\![X]\!]_{\rho'}(w_1, w_2)$, then it suffices to show $\exists v_{1f}, v_{2f}.e_1'[w_1/x] \to^* v_{1f} \wedge w_2 \to^* v_{2f} \wedge \mathcal{V}[\![X]\!]_{\rho'}(v_{1f}, v_{2f})$. Finally, instantiate (5.5) with w_1 and w_2. Then we know $e_1'[w_1/x] \to^* v_{1f}'$, and $e_2'[w_2/x]) \to^* v_{2f}'$, and $\mathcal{V}[\![X]\!]_{\rho'}(v_{1f}', v_{2f}')$. Thus, by the definition of our value interpretation, $(v_{1f}', v_{2f}') \in R$. Now use v_{1f}' for v_{1f} and w_2 for v_{2f} in our goal. Then we must show $(v_{1f'}, w_2) \in R$. To show this, we need to do some arguing about the evaluation of $(\gamma_2(e)\ \_)\ w_2$. We have

$$(\gamma_2(e)\ \_)\ w_2 \to^* (\Lambda\ e_2\ \_)\ w_2 \to e_2\ w_2 \to^* (\lambda x.\ e_2')\ w_2 \to e_2'[w_2/x] \to^* v_{2f}'$$

Since $\bullet \mid \bullet \vdash \gamma_2(e) : \forall X.\ X \to X$ then by theorem 22 we may conclude $(\gamma_2(e)\ \_)\ w_2 \to^* w_2$. Thus by determinacy of System F (specifically, corollary 6.1), we have that $v_{2f}' = w_2$. Since we know $(v_{1f}', v_{2f}') \in R$, then it follows that $(v_{1f}', w_2) \in R$, which was what we had to show. $\square$

5.2 Empty Type

A type is empty, if there are no values with that type. In this section we shall show that $\forall X.\ X$ is an empty type. We do this by showing that there are no expressions of that type, and hence also no values.

Theorem 24. $\nexists e. \bullet \mid \bullet \vdash e : \forall X.\ X$

Proof. Assume for the sake of contradiction that $\exists e. \bullet \mid \bullet \vdash e : \forall X.\ X$. Then, by the fundamental theorem (theorem 21), we know that $\bullet \mid \bullet \vdash e \approx^{LR} e : \forall X.\ X$, which means that $\forall \rho \in \mathcal{D}[\![\bullet]\!], \gamma \in \mathcal{G}[\![\bullet]\!]_\rho.\mathcal{E}[\![\forall X.\ X]\!]_\rho(\gamma_1(e), \gamma_2(e))$. Here, pick the empty relational substitutions, so that $\mathcal{E}[\![\forall X.\ X]\!]_\emptyset(e, e)$. This means that $\exists v_1, v_2.e \to^* v_1 \wedge e \to^* v_2 \wedge \mathcal{V}[\![\forall X.\ X]\!]_\emptyset(v_1, v_2)$. From this we may conclude that $e \to^* \Lambda\ e'$, and $\forall \tau_1, \tau_2, R \in Rel[\tau_1, \tau_2].\mathcal{E}[\![X]\!]_{[X \mapsto (\tau_1, \tau_2, R)]}(e', e')$. Instantiate this with any two types (for example $\mathbb{Z}$), and the empty relation, $R = \emptyset$. Then we have $\mathcal{E}[\![X]\!]_{[X\mapsto(\mathbb{Z},\mathbb{Z},\emptyset)]}(e', e')$, from which we know $\exists v_1, v_2. e' \to^* v_1 \wedge e' \to^* v_2 \wedge \mathcal{V}[\![X]\!]_{[X\mapsto(\mathbb{Z},\mathbb{Z},\emptyset)]}(v_1, v_2)$. By the definition of the value interpretation for type variables, the last part gives us $(v_1, v_2) \in \emptyset$, which is a contradiction. $\square$

5.3 Idempotency

For the next three theorems, we shall introduce let-expressions, which will just be syntactic sugar for a function application: $\mathsf{let}\ x = e\ \mathsf{in}\ e' \triangleq (\lambda x.\ e')\ e$.

Idempotency in general is the idea that doing something multiple times is equivalent to doing it once. In our theorem below, it corresponds to the fact that evaluating the same expression multiple times does not yield different behaviour compared to just evaluating it once, and using the value you get in its place.

Theorem 25. *If* $\Xi \mid \Gamma \vdash e : \tau$ *then* $\Xi \mid \Gamma \vdash \mathsf{let}\ x = e\ \mathsf{in}\ (x, x) \approx^{ctx} (e, e) : \tau \times \tau$

Proof. By theorem 20, it suffices to show logical relatedness. So assume $\Xi \mid \Gamma \vdash e : \tau$, then we will show $\Xi \mid \Gamma \vdash \mathsf{let}\ x = e\ \mathsf{in}\ (x, x) \approx^{LR} (e, e) : \tau \times \tau$. By definition of LR, we must show two things. First that both expressions are well-typed under Ξ and Γ. This follows fairly easily from the assumption and by applying the typing rules, so we will not explain it here. Second that, when the expressions and types are closed, they are in the expression interpretation. So assume some $\rho \in \mathcal{D}[\![\Xi]\!]$ and $\gamma \in \mathcal{G}[\![\Gamma]\!]_\rho$. Then we must show $\mathcal{E}[\![\tau \times \tau]\!]_\rho(\gamma_1(\mathsf{let}\ x = e\ \mathsf{in}\ (x, x)), \gamma_2((e, e)))$, which by the definition of the expression interpretation means we have to show:

$$\bullet \mid \bullet \vdash \mathsf{let}\ x = \gamma_1(e)\ \mathsf{in}\ (x, x) : \rho_1(\tau \times \tau) \tag{5.6}$$
$$\bullet \mid \bullet \vdash (\gamma_2(e), \gamma_2(e)) : \rho_2(\tau \times \tau) \tag{5.7}$$
$$\exists v_1, v_2. \mathsf{let}\ x = \gamma_1(e)\ \mathsf{in}\ (x, x) \to^* v_1 \wedge (\gamma_2(e), \gamma_2(e)) \to^* v_2 \wedge \mathcal{V}[\![\tau \times \tau]\!]_\rho(v_1, v_2) \tag{5.8}$$

From the initial assumption and the fundamental theorem (theorem 21) we have that $\Xi \mid \Gamma \vdash e \approx^{LR} e : \tau$, which, when we instantiate with ρ and γ, gives us

$$\bullet \mid \bullet \vdash \gamma_1(e) : \rho_1(\tau) \tag{5.9}$$
$$\bullet \mid \bullet \vdash \gamma_2(e) : \rho_2(\tau) \tag{5.10}$$
$$\exists v_1', v_2'. \gamma_1(e) \to^* v_1' \wedge \gamma_2(e) \to^* v_2' \wedge \mathcal{V}[\![\tau]\!]_\rho(v_1', v_2') \tag{5.11}$$

From (5.9) and (5.10), we may prove the first two parts of the goal (5.6) and (5.7). So let's turn our attention the last part (5.8). We use (v_1', v_1') for v_1 and (v_2', v_2') for v_2. The first evaluation holds by:

$$\mathsf{let}\ x = \gamma_1(e)\ \mathsf{in}\ (x, x) \equiv (\lambda x.\ (x, x))\ \gamma_1(e) \to^* (\lambda x.\ (x, x))\ v_1' \to (x, x)\left[v_1'/x\right] \equiv (v_1', v_1') \tag{5.12}$$

The first equivalence simply de-sugars the let expression. The next "$\to^*$" follows from (5.11). The next step is a β-reduction, and the final equivalence is just performing the substitution. Now, the second evaluation holds by:

$$(\gamma_2(e), \gamma_2(e)) \to^* (v_2', \gamma_2(e)) \to^* (v_2', v_2') \tag{5.13}$$

Both "$\to^*$" follows from (5.11). So only $\mathcal{V}[\![\tau \times \tau]\!]_\rho((v_1', v_1'), (v_2', v_2'))$ remains to be shown. By the definition of the value interpretation for product type, it suffices to show well-typedness and $\mathcal{V}[\![\tau]\!]_\rho(v_1', v_2') \wedge \mathcal{V}[\![\tau]\!]_\rho(v_1', v_2')$. Both of which follows from the last part of (5.11). □

5.4 Commutativity

In short, this commutativity theorem states that the order of evaluation of expressions in pairs is unimportant. Note the order of evaluation in the theorem below. In the left expression, e_2 is reduced first. And in the right, e_1 is reduced first.

Theorem 26. *If* $\Xi \mid \Gamma \vdash e_1 : \tau_1$ *and* $\Xi \mid \Gamma \vdash e_2 : \tau_2$ *then*
$\Xi \mid \Gamma \vdash \mathsf{let}\ x = e_2\ \mathsf{in}\ (e_1, x) \approx^{ctx} (e_1, e_2) : \tau_1 \times \tau_2$

Proof. As in the previous proofs, we shall show this by showing the they are logically related. So assume $\Xi \mid \Gamma \vdash e_1 : \tau_1$ and $\Xi \mid \Gamma \vdash e_2 : \tau_2$. We will ignore showing well-typedness here, so assume some $\rho \in \mathcal{D}[\![\Xi]\!]$ and $\gamma \in \mathcal{G}[\![\Gamma]\!]_\rho$, then we must show $\mathcal{E}[\![\tau_1 \times \tau_2]\!]_\rho(\gamma_1(\mathsf{let}\ x = e_2\ \mathsf{in}\ (e_1, x)), \gamma_2((e_1, e_2)))$, which corresponds to showing:

$$\exists v_1, v_2 . \mathsf{let}\ x = \gamma_1(e_2)\ \mathsf{in}\ (\gamma_1(e_1), x) \to^* v_1 \wedge (\gamma_2(e_1), \gamma_2(e_2)) \to^* v_2 \wedge \\ \mathcal{V}[\![\tau_1 \times \tau_2]\!]_\rho(v_1, v_2) \tag{5.14}$$

From the initial assumptions and the fundamental theorem (theorem 21) we have that $\Xi \mid \Gamma \vdash e_1 \approx^{LR} e_1 : \tau_1$ and $\Xi \mid \Gamma \vdash e_2 \approx^{LR} e_2 : \tau_2$, which, when we instantiate with ρ and γ, gives us:

$$\exists v_{1f_1}, v_{1f_2} . \gamma_1(e_1) \to^* v_{1f_1} \wedge \gamma_2(e_1) \to^* v_{1f_2} \wedge \mathcal{V}[\![\tau_1]\!]_\rho(v_{1f_1}, v_{1f_2}) \tag{5.15}$$
$$\exists v_{2f_1}, v_{2f_2} . \gamma_1(e_2) \to^* v_{2f_1} \wedge \gamma_2(e_2) \to^* v_{2f_2} \wedge \mathcal{V}[\![\tau_2]\!]_\rho(v_{2f_1}, v_{2f_2}) \tag{5.16}$$

Now we will instantiate our goal (5.14). Use (v_{1f_1}, v_{2f_1}) for v_1 and (v_{1f_2}, v_{2f_2}) for v_2. Thus, we must show:

$$\mathsf{let}\ x = \gamma_1(e_2)\ \mathsf{in}\ (\gamma_1(e_1), x) \to^* (v_{1f_1}, v_{2f_1}) \tag{5.17}$$
$$(\gamma_2(e_1), \gamma_2(e_2)) \to^* (v_{1f_2}, v_{2f_2}) \tag{5.18}$$
$$\mathcal{V}[\![\tau_1 \times \tau_2]\!]_\rho((v_{1f_1}, v_{2f_1}), (v_{1f_2}, v_{2f_2})) \tag{5.19}$$

The first evaluation (5.17) holds by:

$$\mathsf{let}\ x = \gamma_1(e_2)\ \mathsf{in}\ (\gamma_1(e_1), x) \equiv (\lambda x.\ (\gamma_1(e_1), x))\ \gamma_1(e_2) \to^* \\ (\lambda x.\ (\gamma_1(e_1), x))\ v_{2f_1} \to (\gamma_1(e_1), x)[v_{2f_1}/x] \equiv (\gamma_1(e_1), v_{2f_1}) \to^* (v_{1f_1}, v_{2f_1})$$

The second one (5.18) holds by:

$$(\gamma_2(e_1), \gamma_2(e_2)) \to^* (v_{1f_2}, \gamma_2(e_2)) \to^* (v_{1f_2}, v_{2f_2})$$

Finally, (5.19) follows from the last parts of (5.15) and (5.16). □

5.5 Lam Hoisting

The final theorem we shall show is one which is useful, were we to write a compiler for System F. In some compilers, there is a phase which performs "hoisting". Hoisting is the act of lifting invariant declarations out of loops or functions. We don't have loops in our language, so the theorem below shows that it is safe to hoist declarations (let-expressions) out of lambda abstractions.

Theorem 27. *If* $\Xi \mid \Gamma \vdash e_1 : \tau$ *and* $\Xi \mid \Gamma, y : \tau, x : \tau_1 \vdash e_2 : \tau_2$ *then* $\Xi \mid \Gamma \vdash \mathsf{let}\ y = e_1\ \mathsf{in}\ \lambda x.\ e_2 \approx^{ctx} \lambda x.\ \mathsf{let}\ y = e_1\ \mathsf{in}\ e_2 : \tau_1 \to \tau_2$

Proof. So assuming

$$\Xi \mid \Gamma \vdash e_1 : \tau \tag{5.20}$$

$$\Xi \mid \Gamma, y : \tau, x : \tau_1 \vdash e_2 : \tau_2 \tag{5.21}$$

it suffices to show $\Xi \mid \Gamma \vdash \mathsf{let}\ y = e_1\ \mathsf{in}\ \lambda x.\ e_2 \approx^{LR} \lambda x.\ \mathsf{let}\ y = e_1\ \mathsf{in}\ e_2 : \tau_1 \to \tau_2$. Using our two assumptions (5.20 and 5.21), along with the typing rules T-lam and T-app, we may conclude:

$$\Xi \mid \Gamma \vdash \mathsf{let}\ y = e_1\ \mathsf{in}\ \lambda x.\ e_2 : \tau_1 \to \tau_2 \tag{5.22}$$

$$\Xi \mid \Gamma \vdash \lambda x.\ \mathsf{let}\ y = e_1\ \mathsf{in}\ e_2 : \tau_1 \to \tau_2 \tag{5.23}$$

So assume some $\rho \in \mathcal{D}[\![\Xi]\!]$ and $\gamma \in \mathcal{G}[\![\Gamma]\!]_\rho$, then we must show $\mathcal{E}[\![\tau_1 \to \tau_2]\!]_\rho(\mathsf{let}\ y = \gamma_1(e_1)\ \mathsf{in}\ \lambda x.\ \gamma_1(e_2), \lambda x.\ \mathsf{let}\ y = \gamma_2(e_1)\ \mathsf{in}\ \gamma_2(e_2))$. The closedness and well-typedness follows from applying the fundamental theorem on the above judgements (5.22 and 5.23), and instantiating with ρ and γ. So what remains to be shown is:

$$\exists v_1, v_2.(\mathsf{let}\ y = \gamma_1(e_1)\ \mathsf{in}\ \lambda x.\ \gamma_1(e_2)) \to^* v_1 \wedge \\ (\lambda x.\ \mathsf{let}\ y = \gamma_2(e_1)\ \mathsf{in}\ \gamma_2(e_2)) \to^* v_2 \wedge \mathcal{V}[\![\tau_1 \to \tau_2]\!]_\rho(v_1, v_2) \tag{5.24}$$

From the fundamental theorem on (5.20) instantiated with ρ and γ, we get

$$\gamma_1(e_1) \to^* v_{y_1} \wedge \gamma_2(e_1) \to^* v_{y_2} \wedge \mathcal{V}[\![\tau]\!]_\rho(v_{y_1}, v_{y_2}) \tag{5.25}$$

for some values v_{y_1}, v_{y_2}. We also get that $\bullet \mid \bullet \vdash \gamma_2(e_1) : \rho_1(\tau)$. In our goal (5.24), we use $\lambda x.\ \gamma_1(e_2)[v_{y_1}/y]$ for v_1 and $\lambda x.\ \textsf{let}\ y = \gamma_2(e_1)\ \textsf{in}\ \gamma_2(e_2)$ for v_2. One may see that the first reduction holds by desugaring the let expression into a function application, then using $\gamma_1(e_1) \to^* v_{y_1}$ from (5.25) and the E-lam-app rule to conclude:

$$(\textsf{let}\ y = \gamma_1(e_1)\ \textsf{in}\ \lambda x.\ \gamma_1(e_2)) \equiv (\lambda y.\ (\lambda x.\ \gamma_1(e_2)))\ \gamma_1(e_1) \to^* \\ (\lambda y.\ (\lambda x.\ \gamma_1(e_2)))\ v_{y_1} \to \lambda x.\ \gamma_1(e_2)[v_{y_1}/y]$$

The second reduction holds trivially, so all we need to show is:

$$\mathcal{V}[\![\tau_1 \to \tau_2]\!]_\rho(\lambda x.\ \gamma_1(e_2)[v_{y_1}/y], \lambda x.\ \textsf{let}\ y = \gamma_2(e_1)\ \textsf{in}\ \gamma_2(e_2)) \tag{5.26}$$

So we assume some w_1, w_2, and $\mathcal{V}[\![\tau_1]\!]_\rho(w_1, w_2)$, and show:

$$\exists v_1, v_2.(\gamma_1(e_2)[v_{y_1}/y])[w_1/x] \to^* v_1 \wedge \\ (\textsf{let}\ y = \gamma_2(e_1)\ \textsf{in}\ \gamma_2(e_2))[w_2/x] \to^* v_2 \wedge \mathcal{V}[\![\tau_2]\!]_\rho(v_1, v_2) \tag{5.27}$$

We now use the fundamental theorem on (5.21) and instantiate it with ρ and $\gamma' = \gamma[y \mapsto (v_{y_1}, v_{y_2})][x \mapsto (w_1, w_2)]$. Note that this is fine as both variables map to pairs of values that are in the value interpretation at their respective types. From this we get:

$$\gamma_1'(e_2) \to^* v_{f_1} \wedge \gamma_2'(e_2) \to^* v_{f_2} \wedge \mathcal{V}[\![\tau_2]\!]_\rho(v_{f_1}, v_{f_2}) \tag{5.28}$$

For some values v_{f_1}, v_{f_2}. Finally, in our goal (5.27), we use v_{f_1} for v_1, and v_{f_2} for v_2. Firstly, $\mathcal{V}[\![\tau_2]\!]_\rho(v_{f_1}, v_{f_2})$ follows directly from (5.28), so let's show that the evaluations hold. The first one holds by:

$$\begin{aligned}
(\gamma_1(e_2)[v_{y_1}/y])[w_1/x] &\equiv \gamma_1[y \mapsto v_{y_1}](e_2)[w_1/x] && \text{(substitution lemma)}\\
&\equiv \gamma_1[y \mapsto v_{y_1}][x \mapsto w_1](e_2) && \text{(substitution lemma)}\\
&\equiv \gamma_1'(e_2)\\
&\to^* v_{f_1} && \text{(from (5.28))}
\end{aligned}$$

The second evaluation holds by:

$$\begin{aligned}
(\textsf{let}\ y = \gamma_2(e_1)\ \textsf{in}\ \gamma_2(e_2))[w_2/x] &\equiv \textsf{let}\ y = \gamma_2(e_1)[w_2/x]\ \textsf{in}\ \gamma_2(e_2)[w_2/x]\\
&\equiv \textsf{let}\ y = \gamma_2(e_1)\ \textsf{in}\ \gamma_2(e_2)[w_2/x]\\
&\to^* \textsf{let}\ y = v_{y_2}\ \textsf{in}\ \gamma_2(e_2)[w_2/x]\\
&\to (\gamma_2(e_2)[w_2/x])[v_{y_2}/y]\\
&\equiv \gamma_2[x \mapsto w_2](e_2)[v_{y_2}/y]\\
&\equiv \gamma_2[x \mapsto w_2][y \mapsto v_{y_2}](e_2)\\
&\equiv \gamma_2'(e_2)\\
&\to^* v_{f_2}
\end{aligned}$$

In the first equivalence, we simply push in the substitution. The next equivalence holds as $\gamma_2(e_1)$ is a closed expression. Next we use the fact that $\gamma_2(e_1) \to^* v_{y_2}$. Following that, the let-expression desugars into a function application where both expressions are values, so we use E-lam-app to take a step. Then we apply the substitution lemma twice, and the last step holds by (5.28). $\square$

Chapter 6

Conclusion and Ideas for Future Work

In this report, we defined contextual equivalence on System F, which states that two well-typed expressions e, e' are contextually equivalent if, when plugged into an arbitrary context C that takes expressions of their common type, and becomes closed at type boolean, $C[e]$ terminates at some value if and only if $C[e']$ terminates at the same value. We also showed how we could think of contextual equivalence as the coarsest relation that was adequate, a congruency relation, and wherein all expressions were well-typed.

We introduced the concept of logical relations, and how it, as a proof technique, works by being a subset of the relation of interest, which in our case was contextual equivalence. The insight we got from the second definition of contextual equivalence then made it clear which properties our logical relations model should have.

We then presented a logical relations model, and showed that it had these properties. Congruency we got from proving compatibility lemmas, of which we showed a few interesting ones, and adequacy we showed using determinism of system F. This then allowed us to conclude that the model was sound with respect to contextual equivalence. The compatibility lemmas also gave rise to the fundamental theorem – a very powerful result, which we could even use to show that System F is a normalising language.

Finally, we showed some examples of how to use the logical relations model to prove interesting theorems. We proved two free theorems; first that an expression of type $\forall X.\ X \to X$ must act as the identity function, and second that the type $\forall X.\ X$ is an empty type. We also showed four contextual equivalences; identity, commutativity, idempotency, and lambda hoisting.

For future work, we could add more functionality to the language such as recursive functions and existential types. System F is a normalising language, and this influenced how we defined contextual equivalence and thus also the logical relations model. Adding recursive functions would mean that our definition of contextual equivalence would change, and the logical relations model we would have to define would become more complex, requiring step-indexing.

Bibliography

[1] Benjamin C. Pierce. *Types and programming languages*. MIT Press, 2002.

[2] Lau Skorstengaard. An introduction to logical relations. *CoRR*, abs/1907.11133, 2019.

[3] Philip Wadler. Theorems for free! In Joseph E. Stoy, editor, *Proceedings of the fourth international conference on Functional programming languages and computer architecture, FPCA 1989, London, UK, September 11-13, 1989*, pages 347–359. ACM, 1989.

Appendix A

Congruence Rules for Contextual Equivalence

$$\textsc{Cng-var}\quad \frac{(x : \tau) \in \Gamma}{\Xi \mid \Gamma \vdash x \approx x : \tau}$$

$$\textsc{Cng-unit}\quad \frac{}{\Xi \mid \Gamma \vdash () \approx () : \mathsf{Unit}}$$

$$\textsc{Cng-int}\quad \frac{}{\Xi \mid \Gamma \vdash \overline{n} \approx \overline{n} : \mathbb{Z}}$$

$$\textsc{Cng-add}\quad \frac{\Xi \mid \Gamma \vdash e_1 \approx e_1' : \mathbb{Z} \qquad \Xi \mid \Gamma \vdash e_2 \approx e_2' : \mathbb{Z}}{\Xi \mid \Gamma \vdash e_1 + e_2 \approx e_1' + e_2' : \mathbb{Z}}$$

$$\textsc{Cng-sub}\quad \frac{\Xi \mid \Gamma \vdash e_1 \approx e_1' : \mathbb{Z} \qquad \Xi \mid \Gamma \vdash e_2 \approx e_2' : \mathbb{Z}}{\Xi \mid \Gamma \vdash e_1 - e_2 \approx e_1' - e_2' : \mathbb{Z}}$$

$$\textsc{Cng-le}\quad \frac{\Xi \mid \Gamma \vdash e_1 \approx e_1' : \mathbb{Z} \qquad \Xi \mid \Gamma \vdash e_2 \approx e_2' : \mathbb{Z}}{\Xi \mid \Gamma \vdash e_1 \leq e_2 \approx e_1' \leq e_2' : \mathbb{B}}$$

$$\textsc{Cng-lt}\quad \frac{\Xi \mid \Gamma \vdash e_1 \approx e_1' : \mathbb{Z} \qquad \Xi \mid \Gamma \vdash e_2 \approx e_2' : \mathbb{Z}}{\Xi \mid \Gamma \vdash e_1 < e_2 \approx e_1' < e_2' : \mathbb{B}}$$

$$\textsc{Cng-eq}\quad \frac{\Xi \mid \Gamma \vdash e_1 \approx e_1' : \mathbb{Z} \qquad \Xi \mid \Gamma \vdash e_2 \approx e_2' : \mathbb{Z}}{\Xi \mid \Gamma \vdash e_1 = e_2 \approx e_1' = e_2' : \mathbb{B}}$$

$$\textsc{Cng-true}\quad \frac{}{\Xi \mid \Gamma \vdash \mathsf{true} \approx \mathsf{true} : \mathbb{B}}$$

$$\textsc{Cng-false}\quad \frac{}{\Xi \mid \Gamma \vdash \mathsf{false} \approx \mathsf{false} : \mathbb{B}}$$

$$\textsc{Cng-if}\quad \frac{\Xi \mid \Gamma \vdash e_1 \approx e_1' : \mathbb{B} \qquad \Xi \mid \Gamma \vdash e_2 \approx e_2' : \tau \qquad \Xi \mid \Gamma \vdash e_3 \approx e_3' : \tau}{\Xi \mid \Gamma \vdash \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 \approx \mathsf{if}\ e_1'\ \mathsf{then}\ e_2'\ \mathsf{else}\ e_3' : \tau}$$

$$\frac{\Xi \mid \Gamma \vdash e_1 \approx e_1' : \tau_1 \qquad \Xi \mid \Gamma \vdash e_2 \approx e_2' : \tau_2}{\Xi \mid \Gamma \vdash (e_1, e_2) \approx (e_1', e_2') : \tau_1 \times \tau_2}\ \textsc{Cng-pair}$$

$$\frac{\Xi \mid \Gamma \vdash e \approx e' : \tau_1 \times \tau_2}{\Xi \mid \Gamma \vdash \mathsf{fst}\ e \approx \mathsf{fst}\ e' : \tau_1}\ \textsc{Cng-fst}$$

$$\frac{\Xi \mid \Gamma \vdash e \approx e' : \tau_1 \times \tau_2}{\Xi \mid \Gamma \vdash \mathsf{snd}\ e \approx \mathsf{snd}\ e' : \tau_2}\ \textsc{Cng-snd}$$

$$\frac{\Xi \mid \Gamma \vdash e \approx e' : \tau_1}{\Xi \mid \Gamma \vdash \mathsf{inj}_1\ e \approx \mathsf{inj}_1\ e' : \tau_1 + \tau_2}\ \textsc{Cng-inj1}$$

$$\frac{\Xi \mid \Gamma \vdash e \approx e' : \tau_2}{\Xi \mid \Gamma \vdash \mathsf{inj}_2\ e \approx \mathsf{inj}_2\ e' : \tau_1 + \tau_2}\ \textsc{Cng-inj2}$$

$$\frac{\Xi \mid \Gamma \vdash e_1 \approx e_1' : \tau_1 + \tau_2 \qquad \Xi \mid \Gamma, x : \tau_1 \vdash e_2 \approx e_2' : \tau \qquad \Xi \mid \Gamma, x : \tau_2 \vdash e_3 \approx e_3' : \tau}{\Xi \mid \Gamma \vdash \mathsf{match}\ e_1\ \mathsf{with}\ \mathsf{inj}_1\ x \Rightarrow e_2 \mid \mathsf{inj}_2\ x \Rightarrow e_3\ \mathsf{end} \approx \mathsf{match}\ e_1'\ \mathsf{with}\ \mathsf{inj}_1\ x \Rightarrow e_2' \mid \mathsf{inj}_2\ x \Rightarrow e_3'\ \mathsf{end} : \tau}\ \textsc{Cng-match}$$

$$\frac{\Xi \mid \Gamma, x : \tau_1 \vdash e \approx e' : \tau_2}{\Xi \mid \Gamma \vdash \lambda x.\ e \approx \lambda x.\ e' : \tau_1 \to \tau_2}\ \textsc{Cng-lam}$$

$$\frac{\Xi \mid \Gamma \vdash e_1 \approx e_1' : \tau_1 \to \tau_2 \qquad \Xi \mid \Gamma \vdash e_2 \approx e_2' : \tau_1}{\Xi \mid \Gamma \vdash e_1\ e_2 \approx e_1'\ e_2' : \tau_2}\ \textsc{Cng-app}$$

$$\frac{\Xi, X \mid \Gamma \vdash e \approx e' : \tau}{\Xi \mid \Gamma \vdash \Lambda\ e \approx \Lambda\ e' : \forall X.\ \tau}\ \textsc{Cng-TLam}$$

$$\frac{\Xi \mid \Gamma \vdash e \approx e' : \forall X.\ \tau}{\Xi \mid \Gamma \vdash e\ \_ \approx e'\ \_ : \tau[\tau'/X]}\ \textsc{Cng-TApp}$$

Appendix B

Additional Results for the Logical Relations Model

Lemma 28. $X \notin \text{dom}(\rho) \wedge X \notin \text{free}(\tau) \implies \forall \tau_1, \tau_2, R \in Rel[\tau_1, \tau_2].\mathcal{V}[\![\tau]\!]_\rho(v_1, v_2) \iff \mathcal{V}[\![\tau]\!]_{\rho[X \mapsto (\tau_1, \tau_2, R)]}(v_1, v_2)$

Proof. The proof won't be carried out here, but it follows by induction on τ. $\square$

Corollary 28.1. $X \notin \text{dom}(\rho) \wedge X \notin \text{free}(\Gamma) \implies \forall \tau_1, \tau_2, R \in Rel[\tau_1, \tau_2].\mathcal{G}[\![\Gamma]\!]_\rho = \mathcal{G}[\![\Gamma]\!]_{\rho[X \mapsto (\tau_1, \tau_2, R)]}$

Proof. Assuming $X \notin \text{dom}(\rho)$, $X \notin \text{free}(\Gamma)$, and some τ_1, τ_2, and $R \in Rel[\tau_1, \tau_2]$, we must show $\mathcal{G}[\![\Gamma]\!]_\rho = \mathcal{G}[\![\Gamma]\!]_{\rho[X \mapsto (\tau_1, \tau_2, R)]}$. We proceed by induction on Γ. The base case is trivial: $\mathcal{G}[\![\bullet]\!]_\rho = \mathcal{G}[\![\bullet]\!]_{\rho[X \mapsto (\tau_1, \tau_2, R)]} = \{\emptyset\}$. So let's show the inductive step. Our induction hypothesis is $\mathcal{G}[\![\Gamma]\!]_\rho = \mathcal{G}[\![\Gamma]\!]_{\rho[X \mapsto (\tau_1, \tau_2, R)]}$, and we must show $\mathcal{G}[\![\Gamma, x : \tau]\!]_\rho = \mathcal{G}[\![\Gamma, x : \tau]\!]_{\rho[X \mapsto (\tau_1, \tau_2, R)]}$, which is equivalent to showing $\gamma \in \mathcal{G}[\![\Gamma, x : \tau]\!]_\rho \iff \gamma \in \mathcal{G}[\![\Gamma, x : \tau]\!]_{\rho[X \mapsto (\tau_1, \tau_2, R)]}$, for all γ. Both directions of the double implication are similar, so we show just the " $\implies$ " direction. So assuming $\gamma \in \mathcal{G}[\![\Gamma, x : \tau]\!]_\rho$, we must show $\gamma \in \mathcal{G}[\![\Gamma, x : \tau]\!]_{\rho[X \mapsto (\tau_1, \tau_2, R)]}$. From $\gamma \in \mathcal{G}[\![\Gamma, x : \tau]\!]_\rho$ we may conclude that $\gamma = \gamma'[x \mapsto (v_1, v_2)]$, and that $\gamma' \in \mathcal{G}[\![\Gamma]\!]_\rho$ and $\mathcal{V}[\![\tau]\!]_\rho(v_1, v_2)$. From this and lemma 28 we know that $\mathcal{V}[\![\tau]\!]_{\rho[X \mapsto (\tau_1, \tau_2, R)]}(v_1, v_2)$, and from our induction hypothesis, we get $\gamma' \in \mathcal{G}[\![\Gamma]\!]_{\rho[X \mapsto (\tau_1, \tau_2, R)]}$. So by the definition of $\mathcal{G}$, we may conclude $\gamma \in \mathcal{G}[\![\Gamma, x : \tau]\!]_{\rho[X \mapsto (\tau_1, \tau_2, R)]}$. $\square$

Lemma 29 (Substitution lemma LR)**.** *If* $\Xi \mid \Gamma \vdash e : \tau$ *and* $\rho \in \mathcal{D}[\![\Xi]\!]$ *and* $\gamma \in \mathcal{G}[\![\Gamma]\!]_\rho$*, then* $\bullet \mid \bullet \vdash \gamma_1(e) : \rho_1(\tau)$ *and* $\bullet \mid \bullet \vdash \gamma_2(e) : \rho_2(\tau)$

Proof. This follows fairly easily from the fundamental theorem: by the fundamental theorem on $\Xi \mid \Gamma \vdash e : \tau$, we have $\forall \rho \in \mathcal{D}[\![\Xi]\!], \gamma \in \mathcal{G}[\![\Gamma]\!]_\rho.\mathcal{E}[\![\tau]\!]_\rho(\gamma_1(e), \gamma_2(e))$. Using our ρ and γ, we get $\mathcal{E}[\![\tau]\!]_\rho(\gamma_1(e), \gamma_2(e))$. By definition of the expression interpretation, this means that $\bullet \mid \bullet \vdash \gamma_1(e) : \rho_1(\tau)$ and $\bullet \mid \bullet \vdash \gamma_2(e) : \rho_2(\tau)$, which was what we wanted. $\square$

Appendix C

Additional Compatibility Lemmas

Lemma 30 (Cmpt-match)**.**

$$\frac{\Xi \mid \Gamma \vdash e_1 \approx^{LR} e_1' : \tau_1 + \tau_2 \qquad \Xi \mid \Gamma, x : \tau_1 \vdash e_2 \approx^{LR} e_2' : \tau \qquad \Xi \mid \Gamma, x : \tau_2 \vdash e_3 \approx^{LR} e_3' : \tau}{\Xi \mid \Gamma \vdash \mathsf{match}\ e_1\ \mathsf{with}\ \mathsf{inj}_1\ x \Rightarrow e_2 \mid \mathsf{inj}_2\ x \Rightarrow e_3\ \mathsf{end} \approx^{LR} \mathsf{match}\ e_1'\ \mathsf{with}\ \mathsf{inj}_1\ x \Rightarrow e_2' \mid \mathsf{inj}_2\ x \Rightarrow e_3'\ \mathsf{end} : \tau}$$

Proof. First, we introduce the following notation: $e = \mathsf{match}\ e_1\ \mathsf{with}\ \mathsf{inj}_1\ x \Rightarrow e_2 \mid \mathsf{inj}_2\ x \Rightarrow e_3\ \mathsf{end}$, and $e' = \mathsf{match}\ e_1'\ \mathsf{with}\ \mathsf{inj}_1\ x \Rightarrow e_2' \mid \mathsf{inj}_2\ x \Rightarrow e_3'\ \mathsf{end}$. So we assume

$$\Xi \mid \Gamma \vdash e_1 \approx^{LR} e_1' : \tau_1 + \tau_2 \tag{C.1}$$

$$\Xi \mid \Gamma, x : \tau_1 \vdash e_2 \approx^{LR} e_2' : \tau \tag{C.2}$$

$$\Xi \mid \Gamma, x : \tau_2 \vdash e_3 \approx^{LR} e_3' : \tau \tag{C.3}$$

and must show: $\forall \rho \in \mathcal{D}[\![\Xi]\!], \gamma \in \mathcal{G}[\![\Gamma]\!]_\rho . \mathcal{E}[\![\tau]\!]_\rho(\gamma_1(e), \gamma_2(e'))$. So assume some $\rho \in \mathcal{D}[\![\Xi]\!]$, and $\gamma \in \mathcal{G}[\![\Gamma]\!]_\rho$, then we must show $\mathcal{E}[\![\tau]\!]_\rho(\gamma_1(e), \gamma_2(e'))$. We again introduce the following notation $f = \mathsf{match}\ \gamma_1(e_1)\ \mathsf{with}\ \mathsf{inj}_1\ x \Rightarrow \gamma_1(e_2) \mid \mathsf{inj}_2\ x \Rightarrow \gamma_1(e_3)\ \mathsf{end}$, and $f' = \mathsf{match}\ \gamma_2(e_1')\ \mathsf{with}\ \mathsf{inj}_1\ x \Rightarrow \gamma_2(e_2') \mid \mathsf{inj}_2\ x \Rightarrow \gamma_2(e_3')\ \mathsf{end}$, and note that $\gamma_1(e) = f$ and $\gamma_2(e') = f'$. Thus, we must show: $\mathcal{E}[\![\tau]\!]_\rho(f, f')$, which by definition corresponds to showing

$$\exists v_1, v_2 . f \to^* v_1 \wedge f' \to^* v_2 \wedge \mathcal{V}[\![\tau]\!]_\rho(v_1, v_2) \tag{C.4}$$

By C.1 instantiated with ρ, γ, we get

$$\exists v_1, v_2 . \gamma_1(e_1) \to^* v_1 \wedge \gamma_2(e_1') \to^* v_2 \wedge \mathcal{V}[\![\tau_1 + \tau_2]\!]_\rho(v_1, v_2) \tag{C.5}$$

Let's instantiate the values in (C.5) as v_{1_1}' and v_{1_2}', so that we know $\gamma_1(e_1) \to^* v_{1_1}' \wedge \gamma_2(e_1') \to^* v_{1_2}' \wedge \mathcal{V}[\![\tau_1 + \tau_2]\!]_\rho(v_{1_1}', v_{1_2}')$. By definition of the value interpretation, this gives us $(v_{1_1}', v_{1_2}') \in \{(\mathsf{inj}_1\ w_1, \mathsf{inj}_1\ w_2) \mid \mathcal{V}[\![\tau_1]\!]_\rho(w_1, w_2)\} \vee$
$(v_{1_1}', v_{1_2}') \in \{(\mathsf{inj}_2\ w_1, \mathsf{inj}_2\ w_2) \mid \mathcal{V}[\![\tau_2]\!]_\rho(w_1, w_2)\}$. We do case distinction on the

"$\vee$", but show only the first case as they are similar. So assume $(v'_{1_1}, v'_{1_2}) \in \{(\mathsf{inj}_1\ w_1, \mathsf{inj}_1\ w_2) \mid \mathcal{V}[\![\tau_1]\!]_\rho(w_1, w_2)\}$. This means that $v'_{1_1} = \mathsf{inj}_1\ w_1$ and $v'_{1_2} = \mathsf{inj}_1\ w_2$ for some w_1, w_2, and $\mathcal{V}[\![\tau_1]\!]_\rho(w_1, w_2)$. Now, from C.2 instantiated with $\rho, \gamma' = \gamma[x \mapsto (w_1, w_2)]$ we get

$$\exists v_1, v_2. \gamma'_1(e_2) \to^* v_1 \wedge \gamma'_2(e'_2) \to^* v_2 \wedge \mathcal{V}[\![\tau]\!]_\rho(v_1, v_2) \tag{C.6}$$

If we instantiate the values in (C.6) as v'_{2_1} and v'_{2_2}, then we know that $\gamma'_1(e_2) \to^* v'_{2_1} \wedge \gamma'_2(e'_2) \to^* v'_{2_2} \wedge \mathcal{V}[\![\tau]\!]_\rho(v'_{2_1}, v'_{2_2})$. By the substitution lemma (lemma 3), we then know:

$$\gamma_1(e_2)[w_1/x] \to^* v'_{2_1} \tag{C.7}$$

$$\gamma_2(e'_2)[w_2/x] \to^* v'_{2_2} \tag{C.8}$$

Now let us turn to proving the goal, (C.4). We use v'_{2_1} for v_1 and v'_{2_2} for v_2. Thus, we must show $f \to^* v'_{2_1} \wedge f' \to^* v'_{2_2} \wedge \mathcal{V}[\![\tau]\!]_\rho(v'_{2_1}, v'_{2_2})$. We have the following evaluations:
$f \to^* \mathsf{match}\ \mathsf{inj}_1\ w_1\ \mathsf{with}\ \mathsf{inj}_1\ x \Rightarrow \gamma_1(e_2) \mid \mathsf{inj}_2\ x \Rightarrow \gamma_1(e_3)\ \mathsf{end} \to \gamma_1(e_2)[w_1/x] \to^* v'_{2_1}$, and
$f' \to^* \mathsf{match}\ \mathsf{inj}_1\ w_2\ \mathsf{with}\ \mathsf{inj}_1\ x \Rightarrow \gamma_2(e'_2) \mid \mathsf{inj}_2\ x \Rightarrow \gamma_2(e'_3)\ \mathsf{end} \to \gamma_2(e'_2)[w_2/x] \to^* v'_{2_2}$.
In both evaluations, the first "$\to^*$" follows from the instantiation of (C.5), and the fact that $v'_{1_1} = \mathsf{inj}_1\ w_1$ and $v'_{1_2} = \mathsf{inj}_1\ w_2$. The next "$\to$" holds by E-match-inj1. And finally, the last "$\to^*$" holds by (C.7) and (C.8). Thus the first two parts of the goal are satisfied. The last part of the goal we get from the instantiation of (C.6). $\square$

Lemma 31 (Cmpt-app). $\dfrac{\Xi \mid \Gamma \vdash e_1 \approx^{LR} e'_1 : \tau_1 \to \tau_2 \qquad \Xi \mid \Gamma \vdash e_2 \approx^{LR} e'_2 : \tau_1}{\Xi \mid \Gamma \vdash e_1\ e_2 \approx^{LR} e'_1\ e'_2 : \tau_2}$

Proof. So we assume

$$\Xi \mid \Gamma \vdash e_1 \approx^{LR} e'_1 : \tau_1 \to \tau_2 \tag{C.9}$$

$$\Xi \mid \Gamma \vdash e_2 \approx^{LR} e'_2 : \tau_1 \tag{C.10}$$

Then we must show $\forall \rho \in \mathcal{D}[\![\Xi]\!], \gamma \in \mathcal{G}[\![\Gamma]\!]_\rho. \mathcal{E}[\![\tau_2]\!]_\rho(\gamma_1(e_1\ e_2), \gamma_2(e'_1\ e'_2))$. So we assume some $\rho \in \mathcal{D}[\![\Xi]\!], \gamma \in \mathcal{G}[\![\Gamma]\!]_\rho$, and our goal now is $\mathcal{E}[\![\tau_2]\!]_\rho(\gamma_1(e_1\ e_2), \gamma_2(e'_1\ e'_2)) \equiv \mathcal{E}[\![\tau_2]\!]_\rho(\gamma_1(e_1)\ \gamma_1(e_2), \gamma_2(e'_1)\ \gamma_2(e'_2))$, which by definition means we have to show

$$\exists v_1, v_2. \gamma_1(e_1)\ \gamma_1(e_2) \to^* v_1 \wedge \gamma_2(e'_1)\ \gamma_2(e'_2) \to^* v_2 \wedge \mathcal{V}[\![\tau_2]\!]_\rho(v_1, v_2) \tag{C.11}$$

We instantiate both (C.9) and (C.10) with ρ, γ, and expand the definitions to get

$$\exists v'_1, v'_2. \gamma_1(e_1) \to^* v'_1 \wedge \gamma_2(e'_1) \to^* v'_2 \wedge \mathcal{V}[\![\tau_1 \to \tau_2]\!]_\rho(v'_1, v'_2) \tag{C.12}$$

$$\exists v''_1, v''_2. \gamma_1(e_2) \to^* v''_1 \wedge \gamma_2(e'_2) \to^* v''_2 \wedge \mathcal{V}[\![\tau_1]\!]_\rho(v''_1, v''_2) \tag{C.13}$$

Instantiating all quantifications with the same name they quantify over, we get from the definition of the value interpretation for function-type that $v_1' = \lambda x.\ e_1'$ and $v_2' = \lambda x.\ e_2'$, for some e_1' and e_2'. Further,

$$\forall w_1, w_2. \mathcal{V}[\![\tau_1]\!]_\rho(w_1, w_2) \implies \mathcal{E}[\![\tau_2]\!]_\rho(e_1'[w_1/x], e_2'[w_2/x]) \tag{C.14}$$

We instantiate this with v_1'' and v_2''. The antecedent is satisfied by the last part of (C.13), so we know $\mathcal{E}[\![\tau_2]\!]_\rho(e_1'[v_1''/x], e_2'[v_2''/x])$, which by definition means

$$\exists v_{f_1}, v_{f_2}. e_1'\left[v_1''/x\right] \to^* v_{f_1} \wedge e_2'\left[v_2''/x\right] \to^* v_{f_2} \wedge \mathcal{V}[\![\tau_2]\!]_\rho(v_{f_1}, v_{f_2}) \tag{C.15}$$

Now we turn to showing the goal, (C.11). We use v_{f_1} for v_1 and v_{f_2} for v_2, so that we must show: $\gamma_1(e_1)\ \gamma_1(e_2) \to^* v_{f_1} \wedge \gamma_2(e_1')\ \gamma_2(e_2') \to^* v_{f_2} \wedge \mathcal{V}[\![\tau_2]\!]_\rho(v_{f_1}, v_{f_2})$. We have the following evaluations:

$$\gamma_1(e_1)\ \gamma_1(e_2) \to^* \lambda x.\ e_1'\ \gamma_1(e_2) \to^* \lambda x.\ e_1'\ v_1'' \to e_1'\left[v_1''/x\right] \to^* v_{f_1} \tag{C.16}$$

$$\gamma_2(e_1')\ \gamma_2(e_2') \to^* \lambda x.\ e_2'\ \gamma_2(e_2') \to^* \lambda x.\ e_2'\ v_2'' \to e_2'\left[v_2''/x\right] \to^* v_{f_2} \tag{C.17}$$

Here, in both evaluations, the first "$\to^*$" follows by (C.12) and the fact that $v_1' = \lambda x.\ e_1'$ and $v_2' = \lambda x.\ e_2'$. The second "$\to^*$" follows by (C.13). The next "$\to$" follows by rule E-lam-app, and the final "$\to^*$" follows from (C.15). Thus, the first two parts of what we have to show follows from (C.16) and (C.17) respectively, and the last part follows from the last part of (C.15). $\square$