DEPARTMENT OF COMPUTER SCIENCE

Two-Sided Type Systems with Effects

Ram Larg

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Bachelor of Science in the Faculty of Engineering.

Thursday 9th May, 2024

Abstract

We present a two-sided type system with effects. Two-sided systems [1] extend traditional type systems. Traditional type systems provide the guarantee that a well-typed program does not go wrong but have no assurances for an untypable program. Two-sided systems provide two guarantees: well-typed programs do not go wrong, and ill-typed programs do not evaluate. This paper extends the novel two-sided system to track the effects and provide a more fine-grained expression of termination or evaluation.

If types describe the value that a program may evaluate to, effects can describe how the program evaluates to that value [2]. Effects give us refined type information through enriching the types of our system with annotations and making any underlying side-effects from computation explicit.

For example, an annotation representing divergence, $\bot$, could tell us whether a given term can diverge effect specified in the annotated type.

One of the main problems we aim to address includes the semantic meaning of typing judgements in the two-sided paper. On either side of the judgement $\vdash$, the typing $M : A$ and its complement $M : A^c$ can have a different meaning, hidden by implicit effects such as divergence or crashing. Under call-by-value semantics, the meaning of a closed term typed on the left of the judgement $M : A \vdash$, is not equal to the complement of the same term typed on the right $\vdash M : A^c$, leading to specific rules in the system being unsound.

Higher precision in the type effects for a judgement may allow for a more equivalent treatment of complements in both call-by-name and call-by-value evaluation. This dissertation's main contribution is to prove the soundness for the properties of the two-sided system with effects and present possible resolutions to equivalence issue alongside further extensions.

Declaration

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Taught Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, this work is my own work. Work done in collaboration with, or with the assistance of others, is indicated as such. I have identified all material in this dissertation which is not my own work through appropriate referencing and acknowledgement. Where I have quoted or otherwise incorporated material which is the work of others, I have included the source in the references. Any views expressed in the dissertation, other than referenced material, are those of the author.

Ram Larg, Thursday 9th May, 2024

Contents

Ethics Statement

This project did not require ethical review, as determined by my supervisor, Steven Ramsay.

Chapter 1

Introduction

We present a two-sided type system with effects. Two-sided systems [1] extend traditional type systems. Traditional type systems provide the guarantee that a well-typed program does not go wrong but have no assurances for an untypable program. Two-sided systems provide two guarantees: well-typed programs do not go wrong, and ill-typed programs do not evaluate. This paper extends the novel two-sided system to track the effects and provide a more fine-grained expression of termination or evaluation.

If types describe the value that a program may evaluate to, effects can describe how the program evaluates to that value [2]. Effects give us refined type information through enriching the types of our system with annotations and making any underlying side-effects from computation explicit.

For example, an annotation representing divergence, $\bot$, could tell us whether a given term can diverge effect specified in the annotated type.

One of the main problems we aim to address includes the semantic meaning of typing judgements in the two-sided paper. On either side of the judgement $\vdash$, the typing $M : A$ and its complement $M : A^c$ can have a different meaning, hidden by implicit effects such as divergence or crashing. Under call-by-value semantics, the meaning of a closed term typed on the left of the judgement $M : A \vdash$, is not equal to the complement of the same term typed on the right $\vdash M : A^c$, leading to specific rules in the system being unsound such as (CompR).

Higher precision in the type effects for a judgement may allow for a more equivalent treatment of complements in both call-by-name and call-by-value evaluation. This dissertation's main contribution is to prove the soundness for the properties of the two-sided system with effects and present possible resolutions to equivalence issue alongside further extensions.

1.1 Examples

In this section, we will motivate effects by an example of the effect system restoring soundness to (CompR) and possible use cases in the two-sided system.

We illustrate the inference rule for (CompR) below:

$$\frac{\Gamma,\, M : A \vdash}{\Gamma \vdash M : A^c}\,(\mathsf{CompR})$$

The reason why the rule is unsound due to CBV semantics. We discuss the semantics in further detail in the background where we cover call-by-value and call-by-name in Section 2.1.5 and complements in Section 2.1.6, but essentially the difference is: on the right, $\vdash M : A^c$ says M either evaluates to not A value or diverges, whereas on the left $M : A \vdash$ only says that M evaluates to not A value. This means that on the left, M can also diverge or crash.

To resolve this, we have defined effects for the system: divergence $\bot$, crashing $\lightning$ and no effect ε which can be added as an annotation onto the end of a type. We then want to distinguish between annotated and non-annotated types in the system. Non-annotated types are the types of the system as usual (Nat, $A \to B$, etc.), but with no annotation attached and are defined using S and T. Compare these to annotated types, which are defined using A and B, and attach an effect α to a non-annotated type. This gives us the ability to type S_α where S may be a Nat, arrow or pairing with the effect α attached. We use α as a metavariable for any of the 3 effects introduced so far.

Using our current knowledge, we can give annotate the types for simple terms with an effect. One example is the numeral $\underline{2}$. It would make sense if we could give this the type effect $\mathsf{Nat}_{\varepsilon}$ since the term evaluates with no effect. Whereas $\underline{\mathsf{div}}$ is a term that loops infinitely, so $\mathsf{Nat}_{\perp}$ would be appropriate.

Returning to the rule for (CompR), we might be able to formulate a modified rule using effects like so:

$$\frac{\Gamma,\, M : S_{\varepsilon} \vdash}{\Gamma \vdash M : S^{c}{}_{\perp \lightning}}\,(\mathsf{CompR})$$

This rule reads: 'if we refute the term M has type S_{ε} (i.e. does not evaluate to a S)', then we can conclude that 'M must evaluate to a value with type S complement, or diverges, or crashes'. This definition more accurately encompasses what it means to refute $M : A$ on the left. Also, note the complement of ε being $\perp \lightning$. Since effects can be complemented, it may also make sense to define this formally: $\overline{\alpha} = \{\perp, \lightning\} - \alpha$, which we can use to define a more general rule (ECompR)

$$\frac{\Gamma,\, M : S_{\alpha} \vdash}{\Gamma \vdash M : S^{c}{}_{\overline{\alpha}}}\,(\mathsf{ECompR})$$

So, if we had that a term M diverges, crashes, or evaluates to an S^{c} in the premise, we can conclude $\vdash M : S$ where the effect ε is implicit.

The two-sided paper also introduces Ok as a top type to stand for closed all values. With type complements, we can infer that Ok^{c} would be the type that stands for the absence of any value. In the original paper, the semantics for a term such as $\vdash \mathsf{succ}(\underline{\mathsf{id}}) : \mathsf{Ok}^{c}$ would have meant that $\mathsf{succ}(\underline{\mathsf{id}})$ diverges. This is because Ok^{c} is essentially an empty set $\emptyset$ of values, leaving only the possibility to diverge [1]. This clearly unsound, since we know the term $\mathsf{succ}(\underline{\mathsf{id}})$ goes wrong. With effects, we can resolve this discrepancy like so:

$$\frac{\dfrac{\cdots}{\Gamma,\, \mathsf{succ}(\underline{\mathsf{id}}) : \mathsf{Ok}_{\perp} \vdash}}{\Gamma \vdash \mathsf{succ}(\underline{\mathsf{id}}) : \mathsf{Ok}^{c}{}_{\lightning}}\,(\mathsf{ECompR})$$

So $\vdash \mathsf{succ}(\underline{\mathsf{id}}) : \mathsf{Ok}^{c}{}_{\lightning}$ appropriately means that the term crashes.

We can also demonstrate the use of effects for a more complex term:

$$\vdash \mathsf{if}\, M\, \mathsf{then}\, \mathsf{pred}(\underline{\mathsf{id}}\, \mathsf{else}\, \underline{\mathsf{div}}) : \mathsf{Nat}_{\perp \lightning}$$

Depending on whether the guard M is zero or not, the above expression will either crash or reduce infinitely, so we can annotate it appropriately with the effects $\perp \lightning$.

1.2 Contributions

The author was provided with the underlying type system and its semantics as specified in the original two-sided paper [1]. The contributions of this project are:

- Discussion of related effect systems
- Extension of the two-sided system with effects
- Motivating the use of logical relations
- Proof of soundness using a logical predicate to show progress and effect preservation
- Outline of future work to be completed for the effect system

1.3 Outline

We start in Chapter 2 by introducing the basic background for type and effect systems and some related works.

In Chapter 3, we introduce the effect system and its annotated inference rules, including examples for deriving that an annotated effect type is true.

In Chapter 4, we describe the soundness properties we desire the extended system to have. We also highlight issues found in trying to show these properties traditionally, motivating the use of logical relations to prove the system satisfies these conditions using logical predicates $P_{S_{\alpha}}$.

Finally, Chapter 5 elaborates on the potential areas for future work.

Chapter 2

Background

In this chapter, we provide a brief background to the two-sided system, its extensions from PCF, effects, and a discussion of some related works.

2.1 Two-sided type systems

The call-by-value language in the two-sided system is generated by the following definitions, as specified in [1]:

Definition 2.1 (Terms)**.** For a set of term variables x, y, z, consider a functional programming language whose *values*, typically V, W, and whose *terms*, typically M, N, P, Q, are defined by the following grammar:

$$\begin{array}{rcl} V, W & ::= & x \mid \underline{n} \mid (V, W) \mid \lambda x.M \\ M, N, P, Q & ::= & \mathsf{zero} \mid \mathsf{succ}(M) \mid \mathsf{pred}(M) \mid \mathsf{ifz}\, M\, \mathsf{then}\, N\, \mathsf{else}\, P \mid \\ & & \quad x \mid M\, N \mid \lambda x.\, M \mid \mathsf{fix}\, f(x).\, M \mid (M, N) \mid \mathsf{let}\, (x, y) = M\, \mathsf{in}\, N \end{array}$$

Abbreviations We write $\underline{n}$ for *numeral* $\mathsf{succ}^n(\mathsf{zero})$ for $n \in \mathbb{N}$. We write $\mathsf{fix}\, f(x).\, M$ for $\lambda x.\, M$ when $f \notin FV(M)$. We write $\underline{\mathsf{div}}$ as an abbreviation for $(\mathsf{fix}\, f(x).\, f\, x)\, \underline{0}$, and $\underline{\mathsf{id}}$ for $\lambda x.\, x$. Closed values are numerals, pairs, and (fixpoint) abstractions.

Definition 2.2 (Reduction)**.** We reproduce the *evaluation contexts* from [1]. These are defined by the following grammar :

$$\begin{array}{rcl} \mathcal{E}, \mathcal{F} & ::= & \Box \mid (\mathsf{fix}\, f(x).\, M)\, \mathcal{E} \mid \mathcal{E}\, N \mid \mathsf{succ}(\mathcal{E}) \mid \mathsf{pred}(\mathcal{E}) \mid \\ & & \quad (\mathcal{E}, M) \mid (V, \mathcal{E}) \mid \mathsf{let}\, (x, y) = \mathcal{E}\, \mathsf{in}\, N \mid \mathsf{ifz}\, \mathcal{E}\, \mathsf{then}\, N\, \mathsf{else}\, P \end{array}$$

Given a context $\mathcal{E}$, we write $\mathcal{E}[M]$ for the term obtained by replacing the hole $\Box$ by M. The *one-step reduction relation*, written $M \rhd N$, is the binary relation on (possibly open) terms obtained as the closure of the following schema under evaluation contexts.

$$\begin{array}{lrcl} (\mathsf{IfZ}) & \mathsf{ifz}\, \underline{0}\, \mathsf{then}\, N\, \mathsf{else}\, P & \rhd & N \\ (\mathsf{IfS}) & \mathsf{ifz}\, \mathsf{succ}(\underline{n})\, \mathsf{then}\, N\, \mathsf{else}\, P & \rhd & P \\ (\mathsf{Let}) & \mathsf{let}\, (x, y) = (V, W)\, \mathsf{in}\, M & \rhd & M[V/x, W/y] \\ (\mathsf{PredZ}) & \mathsf{pred}(\underline{0}) & \rhd & \underline{0} \\ (\mathsf{PredS}) & \mathsf{pred}(\underline{n+1}) & \rhd & \underline{n} \\ (\mathsf{Fix}\beta) & (\mathsf{fix}\, f(x).\, M)\, V & \rhd & M[V/x, \mathsf{fix}\, f(x).\, M/f] \end{array}$$

The terms on the left-hand side of the above schema are *redexes*, i.e. reducible expressions that are not in normal form. Writing $M \rhd^* N$ means there exists a reduction sequence from N that leads to N. A term M that cannot make a step is said to be in *normal form*. A term with no normal form *diverges* and has an infinite reduction sequence. We write $M \Downarrow V$ just if closed term $M \rhd^* V$, and we say that M *evaluates*. A term is *stuck* just if it is a normal form that is not a value. A term is said to *crash* (*get stuck* or *go wrong*) just if it reduces to a stuck term.

2.1.1 Refutation

Necessity is a new type of function arrow that describes functions that produce a type B 'only if' an A was provided as input ($A \rightarrowtail B$). This is distinct from our usual arrow, where the function ($A \to B$) produces a B 'if' an A was provided as input.

One example use case that was presented in the two-sided paper is $\lambda x.\,(x, \underline{2})$, which can be assigned the type $\mathsf{Nat} \rightarrowtail (\mathsf{Nat} \times \mathsf{Nat})$ [1]. This function requires the necessity type because the function produces a pair of Nat's only if a Nat was provided as input, with the numeral being substituted for the bound x in the body of the abstraction. The same cannot be said for some function $\lambda x.\,(\underline{1}, \underline{2})$. This function already has the property of the body being type $(\mathsf{Nat} \times \mathsf{Nat})$, so it is not necessary that the input was also a Nat for it to have this type. Instead, the latter function could be given the type $\mathsf{Nat} \to (\mathsf{Nat} \times \mathsf{Nat})$ or $A \to (\mathsf{Nat} \times \mathsf{Nat})$ for some type A.

Necessity ($\rightarrowtail$) is required in our system to refute applications, i.e. $MN : B \vdash$. If we want to refute that an application has some type B, we could first assert M is a function of type $A \to B$ and refute N is a A. However, this is not strict enough in the sense that we mentioned in the previous paragraph, where it is possible M has input types that are not just A. Hence, we must first assert that M is type $A \rightarrowtail B$ and then refute that the term N is an A.

Definition 2.3 (Type Assignment)**.** A *typing formula* is a pair $M : A$ of a term M and type A. The term M is the subject of the formula. A *typing judgement* is a pair of finite sets of typings, written $\Gamma \vdash \Delta$. A judgement is provable or derivable according to the rules in Figure 3.1. In every instance, we use (AbsR), (AbnR), (LetR), (LetL1), (LetL2) and (FixR), we require that any bound variables displayed do not occur freely in Γ or Δ.

2.1.2 Removing restrictions

The two-sided system removes the restriction of only allowing variables to be in the assumptions on the left of the turnstile. Judgements such as $M : A \vdash N : B$ can now be read as 'if M has type A, then N has type B'. Multiple terms can also be on the left, which are logically joined by conjunctions, so the judgement

$$M : A,\ N : B \vdash P : C$$

says 'if M has type A **AND** N has type B, then P has type C'. Terms on the right are logically joined by disjunctions, so the term

$$M : A \vdash N : B,\ P : C$$

says 'if M has type A, then either N has type B **OR** then P has type C'. Empty conclusions on the right mean we refute anything on the left. So

$$M : A, N : B \vdash$$

becomes 'M does not have type A **OR** N does not have type B'.

2.1.3 Ill-typedness

The type 'Ok' is introduced as a top type that represents the type of all values. To show that a closed term M has type Ok on the right, it suffices to show that M has some type A since all types A are some non-empty set of values [1]. Otherwise, to show that a closed term M does not reach a value, we refute it is of type Ok, hence $M : \mathsf{Ok} \vdash$. We give this type of term the name *ill-typed*. From this, we can obtain *well-typed programs don't go wrong* and *ill-typed programs don't evaluate* as a definition.

In the context of the original two-sided paper, the meaning of a closed term M being well-typed just implies that it does not get stuck, while being ill-typed means it does not evaluate (i.e. crashes or diverges) rather than being untypable. Despite this, we note that untypable terms that do not go wrong still exist in the language, such as $\Omega = (\lambda x.\,xx)(\lambda x.\,xx)$.

2.1.4 Semantics

Definition 2.4 (Semantics of Types)**.** Let Vals_0 be the set of all closed values. We reproduce the semantics of type from [1] below and interpret types as certain sets of closed, well-behaved terms:

$$\begin{array}{rcl}
[\![\mathsf{Ok}]\!] & = & \mathsf{Vals}_0 \\
[\![\mathsf{Nat}]\!] & = & \{\ \underline{0}, \underline{1}, \underline{2}, \dots\ \} \\
[\![A \times B]\!] & = & \{\ (V, W) \mid V \in [\![A]\!],\ W \in [\![B]\!]\ \} \\
[\![A \to B]\!] & = & \{\ \mathsf{fix}\, f(x).\, M \mid \forall V \in \mathsf{Vals}_0.\, V \in [\![A]\!] \Rightarrow M[V/x][\mathsf{fix}\, f(x).\, M/f] \in \mathcal{T}_\perp[\![B]\!]\ \} \\
[\![A \rightarrowtail B]\!] & = & \{\ \mathsf{fix}\, f(x).\, M \mid \forall V \in \mathsf{Vals}_0.\, M[V/x][\mathsf{fix}\, f(x).\, M/f] \in \mathcal{T}[\![B]\!] \Rightarrow V \in [\![A]\!]\ \} \\
 & & \\
\mathcal{T}[\![A]\!] & = & \{\ M \mid M \Downarrow V,\ V \in [\![A]\!]\ \} \\
\mathcal{T}_\perp[\![A]\!] & = & \{\ M \mid M \in \mathcal{T}[\![A]\!] \vee M \Uparrow\ \}
\end{array}$$

$[\![A]\!]$ is the set of all closed values of type A, $\mathcal{T}[\![A]\!]$ is the set of all terms that would evaluate to a value of type A and $\mathcal{T}_\perp[\![A]\!]$ is the set of all terms that either evaluate to a value of type A or diverge.

2.1.5 Call-by-value and Call-by-name

One of our aims is for a more symmetrical judgement, like those in *call-by-name* (CBN). For the standard right rules to be sound in CBN, we would have to interpret $M : A \vdash$ on the left, as refuting 'M evaluates to A or diverges' rather than just refuting 'M evaluates to an A', as in *call-by value* (CBV). So, the meaning of a formula about the judgement becomes identical in the premise and consequent.

However, in CBN semantics, we cannot deduce useful insights from statements on the left, preventing us from reasoning about practical programs. For example, when refuting an application has a type A on the left $MN : A \vdash$, we only know MN has either evaluated to an A or diverged. In the case MN does not terminate, we cannot reason about the behaviour of M or $M[N/x]$, which may not have been executed at all if the argument N diverges, for example. This means rules such as (AppL) would be unsound in a CBN system. Our effect system attempts to reason about termination more explicitly in hopes of bringing soundness to a more symmetrical system like in CBN.

2.1.6 Complements

Negation is introduced to answer the question of whether it was possible to formalise typings on the left of the judgement of shape $M : A \vdash$ with some complement operator on the type A^c.

Complement Rules

$$(\mathsf{CompL})\ \frac{\Gamma \vdash M : A,\ \Delta}{\Gamma,\ M : A^c \vdash \Delta} \qquad (\mathsf{CompR})\ \frac{\Gamma,\ M : A \vdash \Delta}{\Gamma \vdash M : A^c,\ \Delta}$$

The meaning of the operator $(^c)$ is to essentially take the complement of the type with respect to the set of closed values. The (CompL) rule is sound and comparable to that of our (Dis) rule in 3.1. One example for which the rule should work is for $\mathsf{pred}(\underline{2}) : \mathsf{Ok}^c \vdash$.

However, the rule (CompR) is unsound in CBV semantics. The reason for this is clearer when we unpack the meaning of refuting terms evaluating to a type on the left, as previously mentioned in the introductory Example 1.1. The judgement $M : A \vdash$ refutes that M evaluates to A. This implies that M can either diverge, go wrong, or evaluate to an A^c. However, on the right $\vdash M : A^c$ means only that M evaluates to an A^c or diverges.

The two-sided paper solves this by weakening the semantics on the right of the judgement [1], so typings $M : A$ on the right can also always go wrong. The issue is that the method loses the *well-typed programs don't go wrong* guarantee since typings on the right mean that M can diverge, evaluate or go wrong, giving no assurances about well-typed terms. Despite this, we still have *ill-typed programs don't evaluate* and can prove that programs behave incorrectly.

As noted in the introduction, we take an alternative approach to this issue with our effect system. Our effects would allow us to say that a term $M : A$ can also crash on the right by annotating the type with a $\lightning$ symbol. However, this means we also lose guarantees about well-typed programs on the right. Ideally, we would want to have control over where we specify the effects, meaning we would not have to redefine the semantics on the right completely, but rather allow for the specification of certain effects in scenarios such as the above.

2.2 Previous work on effects

In this section, we cover some relevant literature that highlights prior work in the area of effect systems.

2.2.1 Fluent languages and effect rules

One of the first explorations of a type and effect system was by Gifford and Lucassen [3]. The paper presents fluent languages, which combine the advantages of imperative and functional computation so that operations with side effects from the imperative side are incorporated in such a way that they are declared as part of the type of the program. Side-effects are described in terms of an *effect class*, which determines a sublanguage to which an expression may restricted, which then determines the language facilities an expression may use and what what subroutines it can call.

Every expression has an effect class. While the type of an expression describes the *value* computed, the effect class of an expression describes *how* that value is computed. To ensure the invariants of the sublanguage hold, we are required to enforce constraints onto this sublanguage. This form of verification is called *effect checking*. Although effect checking can be performed dynamically and statically, the focus is on static checking because it combines early error reporting with no runtime overhead. The fluent language introduces a set of type and effect inference rules, showing how an expression's type and effect classes can be inferred from the types and effect classes of its subexpressions. A point of interest presented by this paper is documentation in the form of effect class specifications and enforcing these specifications with an effect-checking system. We find ourselves reproducing a similar form of effect class specification in our table of Inference Rules 3.1 as presented in Chapter 3.

2.2.2 Subtyping and subeffecting

One of the more contemporary works on Type and Effect systems is by Nielson and Nielson [4]. The paper similarly presents type and effect systems as an extension to normal type systems, with effects describing what might occur during program evaluation. Subtyping and subeffecting are introduced as extensions to the underlying type system. The rule for subeffecting or subsumption is reproduced below:

$$\frac{\Gamma \vdash M : S_\alpha \qquad \alpha \subseteq \beta}{\Gamma \vdash M : S_\beta} \tag{2.1}$$

and can be extended with a rule for subtyping:

$$\frac{\Gamma \vdash M : S_\alpha \qquad S \leq T}{\Gamma \vdash M : T_\alpha} \tag{2.2}$$

Where $\alpha \subseteq \beta$ is a partial ordering on effects, and $S \leq T$ is a partial ordering on our non-annotated types. The rule 2.1 tells us that every element of α is also an element of β if $\alpha \subseteq \beta$. Rule 2.2 tells us that every element of S is also an element of T.

Subeffecting generally allows us to 'expand' effects at an earlier point so that they do not conflict with the requirements of the type and effect system [4]. Subtyping is used to 'expand' effects at later points and potentially provides more information on the type and effects of subprograms. Although we do not use these techniques throughout the paper for the proofs written, they may serve as a starting point for optimising the current system as we describe in the conclusion for future work in Section 5.2.

Chapter 3

Type and Effect System

3.1 Effects in a two-sided type system

We introduce a simple two-sided type effect system for this language in Figure 3.1.

Definition 3.1 (Effects)**.** The metavariables α, β or γ, range over *effects* and are defined by the following grammar:

$$\alpha, \beta, \gamma \quad ::= \quad \varepsilon \mid \bot \mid \lightning$$

An effect α, can be *empty* (no effect) ε, *diverge* $\bot$, or *crash* $\lightning$. These describe any side effects that may have taken place during evaluation.

Definition 3.2 (Annotated Types)**.** The *annotated types*, typically A, B and so on, are defined by the following grammar:

$$A, B \quad ::= \quad S_\alpha$$

Definition 3.3 (Non-annotated Types)**.** The *non-annotated types* or just *types* are now modified to typically be S, T, and are defined by the following grammar:

$$S, T \quad ::= \quad \mathsf{Nat} \mid A \to B \mid A \times B \mid A \rightharpoonup B$$

The non-annotated types alone are not expressive enough to account for additional effects obtained from evaluating the body of a function. So, we define two grammars that allow us to reason about effects more clearly by combining annotated types with our non-annotated types, which are the types as usual. The grammar for types is modified so that S and T stand for *types* made from *annotated types* but are not annotated themselves at the top level. We can then reason about the 'inside' of non-annotated types which contain annotated types. Section 3.1.1 gives some examples for clarification.

Definition 3.4 (Effect Composition)**.** The *composition* of effects, written $\oplus$, have the following output.

$$\varepsilon \oplus \varepsilon = \varepsilon$$
$$\bot \oplus \_ = \bot$$
$$\_ \oplus \bot = \bot$$

Where $\_$ denotes any effect.

For application on the right:

$$(\mathsf{AppR})\ \frac{\Gamma \vdash P : (T_\gamma \to S_\alpha)_\beta, \qquad \Gamma \vdash Q : T_\gamma,}{\Gamma \vdash P\,Q : S_{\alpha \oplus \beta \oplus \gamma},}$$

The symbols α, β, and γ are metavariables for an effect. The overall effect is the composition of what is observed from evaluating Q, with effect γ, what is observed from evaluating P, with effect β, and what is obtained from evaluating the body of the function called in P with effect α.

For simplicity, we may also choose to omit the empty effect for variables and constants, so we can write $x : A$, instead of $x : A_\varepsilon$ and $\mathsf{zero} : \mathsf{Nat}$, instead of $\mathsf{zero} : \mathsf{Nat}_\varepsilon$.

Effects which are the same for both the premise and conclusion of a rule can be generalised by writing α for the overall effect.

Structural Rules

$$\text{(Id)}\ \frac{}{\Gamma,\ x : S_\alpha \vdash x : S_\alpha}$$

Right Typing Rules

$$\text{(ZeroR)}\ \frac{}{\Gamma \vdash \mathsf{zero} : \mathsf{Nat}_\varepsilon} \qquad \text{(cR)}\ \frac{\Gamma \vdash M : \mathsf{Nat}_\alpha}{\Gamma \vdash c(M) : \mathsf{Nat}_\alpha}\, c \in \left\{ \begin{array}{l} \mathsf{succ} \\ \mathsf{pred} \end{array} \right\}$$

$$\text{(AbsR)}\ \frac{\Gamma,\ x : T_\delta \vdash N : S_\alpha}{\Gamma \vdash \lambda x.\, N : (T_\delta \to S_\alpha)_\varepsilon} \qquad \text{(AbnR)}\ \frac{\Gamma,\ N : S_\alpha \vdash x : T_\delta}{\Gamma \vdash \lambda x.\, N : (T_\delta \rightarrowtail S_\alpha)_\varepsilon}$$

$$\text{(FixsR)}\ \frac{\Gamma,\ f : (A \to S_\perp)_\varepsilon,\ x : A \vdash M : S_\perp}{\Gamma \vdash \mathsf{fix}\, f(x).\, M : (A \to S_\perp)_\varepsilon} \qquad \text{(FixnR)}\ \frac{\Gamma,\ f : (A \rightarrowtail S_\perp)_\varepsilon,\ M : S_\perp \vdash x : A}{\Gamma \vdash \mathsf{fix}\, f(x).\, M : (A \rightarrowtail S_\perp)_\varepsilon}$$

$$\text{(AppR)}\ \frac{\Gamma \vdash M : (T_\gamma \to S_\alpha)_\beta \qquad \Gamma \vdash N : T_\gamma}{\Gamma \vdash M\, N : S_{\alpha \oplus \beta \oplus \gamma}} \qquad \text{(PairR)}\ \frac{\Gamma \vdash M : S_\alpha \qquad \Gamma \vdash N : T_\beta}{\Gamma \vdash (M,\ N) : (S_\alpha \times T_\beta)_{\alpha \oplus \beta}}$$

$$\text{(LetR)}\ \frac{\begin{array}{l}\Gamma \vdash M : (B \times C)_\alpha, \\ \Gamma,\ x : B,\ y : C \vdash N : S_\alpha,\end{array}}{\Gamma \vdash \mathsf{let}\,(x, y) = M\,\mathsf{in}\,N : S_\alpha} \qquad \text{(IfzR)}\ \frac{\begin{array}{rl}\Gamma \vdash P : S_\beta, \\ \Gamma \vdash M : \mathsf{Nat}_\alpha \qquad \Gamma \vdash N : S_\gamma,\end{array}}{\Gamma \vdash \mathsf{ifz}\, M\, \mathsf{then}\, N\, \mathsf{else}\, P : S_{\alpha \oplus \beta \oplus \gamma}}$$

Left Typing Rules

$$\text{(Dis)}\ \frac{\Gamma \vdash M : S_\alpha}{\Gamma,\ M : T_\varepsilon \vdash}\, S_\alpha \mid\mid T_\varepsilon \qquad \text{(PairL)}\ \frac{\Gamma,\ M_i : A_i \vdash}{\Gamma,\ (M_1,\ M_2) : (A_1 \times A_2)_\varepsilon \vdash}\, i \in \{1, 2\}$$

$$\text{(AppL)}\ \frac{\Gamma \vdash M : (B \rightarrowtail A)_\varepsilon \qquad \Gamma,\ N : B_\varepsilon \vdash}{\Gamma,\ M\, N : A_\varepsilon \vdash} \qquad \text{(cL)}\ \frac{\Gamma,\ M : \mathsf{Nat}_\varepsilon \vdash}{\Gamma,\ c(M) : \mathsf{Nat}_\varepsilon \vdash}\, c \in \left\{ \begin{array}{l} \mathsf{succ} \\ \mathsf{pred} \end{array} \right\}$$

$$\text{(IfzL1)}\ \frac{\Gamma,\ M : \mathsf{Nat}_\varepsilon \vdash}{\Gamma,\ \mathsf{ifz}\, M\, \mathsf{then}\, N\, \mathsf{else}\, P : A_\varepsilon \vdash} \qquad \text{(IfzL2)}\ \frac{\Gamma,\ N : S_\varepsilon \vdash \qquad \Gamma,\ P : S_\varepsilon \vdash}{\Gamma,\ \mathsf{ifz}\, M\, \mathsf{then}\, N\, \mathsf{else}\, P : S_\varepsilon \vdash}$$

$$\text{(LetL1)}\ \frac{\Gamma,\ N : S_\varepsilon \vdash}{\Gamma,\ \mathsf{let}\,(x, y) = M\,\mathsf{in}\,N : S_\varepsilon \vdash} \qquad \text{(LetL2)}\ \frac{\begin{array}{l}\Gamma,\ M : (B_1 \times B_2)_\varepsilon \vdash \\ \Gamma,\ N : S_\varepsilon \vdash x_i : B_i\ \ (\forall i)\end{array}}{\Gamma,\ \mathsf{let}\,(x_1, x_2) = M\,\mathsf{in}\,N : S_\varepsilon \vdash}$$

Figure 3.1: Two-sided type assignment with effects.

Removal of right context Δ We have simplified the system and rules slightly by removing the context on the right. So we are not considering the bound variables on the right of the typing judgement for the rest of the paper, only those in Γ on the left. This helps with simplifying our proofs later on but could be taken into account in later iterations of the system.

Definition 3.5 (Disjointness)**.** We say that the types S and T are *disjoint*, written $S \mid\mid T$, just if either:

1. S is Nat and T is not
2. S is an arrow, and T is not
3. S is a pair, and T is not

3.1.1 Explanation of the effect annotated rules

Id The rule (Id) is the most basic and says if we assumed $x : S_\alpha$, then we can conclude $x : A_\alpha$. If a variable x has some effect α in our assumptions, then we would expect this effect would be carried over into the conclusion as well.

ZeroR The rule (ZeroR) says if we have the numeral zero, we can conclude the term has evaluated with type effect with Nat_ε since it is already a value.

cR The rule (cR) says that if M has type Nat with effect α, then $\mathsf{succ}(M)$ has the same type effect. We know that $\mathsf{succ}(M)$ evaluates to a numeral if we provide it with an evaluating numeral as its input. However, if we provide succ a diverging input such as $\mathsf{Nat}_\perp$, it is clear that the $\mathsf{succ}(M)$ can also diverge with the same effect. So, we can generalise this notion by annotating the Nat type with α in both the premise and conclusion.

AbsR When $f \notin FV(M)$ we can have the rule (AbsR). Since abstractions are values, we can consider that there is 'no effect' from simply defining the function itself [4], so the overall annotated effect of the arrow type is ε. The types T and S carry any effect that may present in the premise into the conclusion.

AbnR When $f \notin FV(M)$ we can have the rule (AbnR). Similar to (AbsR), since abstractions are values, we can consider that there is 'no effect' from simply defining the function itself, so the overall annotated effect of the arrow type is ε. The types T and S similarly carry any effect that may present in the premise into the conclusion.

FixsR The rule (FisxR) differs from (AbsR) in that the function body has a recursive call in f. Hence, we annotate the output type S with $\perp$ to denote that it can possibly diverge. Since this is still an abstraction, the overall effect of the arrow remains as ε.

FixnR The rule (FisxR) differs from (AbnR) in a similar way. The function body contains a recursive call in f. Hence, we annotate the output type S with $\perp$ to denote that it can possibly diverge. Since this is still an abstraction, the overall effect of the arrow remains as ε.

AppR The rule (AppR) uses effect composition and says that the overall type effect results from the outcome of $\alpha \oplus \beta \oplus \gamma$. The result effect will be $\perp$ when any one of α, β or γ are $\perp$. In the case the overall effect is ε, we know the application evaluates fully.

PairR The rule (PairR) says the overall effect is the composition of the type effect from each term in the pair. So if either M or N diverges, we know the pair can also diverge with effect ε. The only case when the pair evaluates fully is when we know both α and β are ε.

LetR The rule (LetR) says that the overall effect is determined by the pair $(B \times C)_\alpha$. If at least one of the terms in the pair diverges, then it is clear that $\perp$ will be passed along, and the resulting effect is that S can also diverge.

IfzR The rule (IfzR) says that the overall effect is the result of the composition $\alpha \oplus \beta \oplus \gamma$. The result effect will be $\bot$ when any one of α, β or γ are $\bot$. Note that if the guard M diverges, then we assume the whole term diverges since N or P might never get evaluated. In the case the overall effect is ε, we know the application evaluates fully.

Left rules The meaning of the left rules are explained comprehensively in [1], so we will not repeat the semantics here. In CBV evaluation, typings $M : A$ on the left are interpreted as M evaluates to A, so most of the rules appear to have the same effect annotation of ε.

Dis The rule (Dis) is similar to the rule (CompL) introduced in background Section 2.1.6. The rule allows for refuting that a term M evaluates to a T by affirming that M evaluates to some disjoint type S. Notice that since we are refuting an evaluation with effect ε, the premise says that M can also diverge $\bot$ or crash.

3.2 Examples

In this section, the following examples help demonstrate how certain effect properties can be proven. Building the premises using the rules we defined in Figure 3.1, we are able to prove the conclusion and gain insights about effects in our assumptions.

Example 3.1. $\vdash \mathsf{ifz}\, y\, \mathsf{then}\, (\lambda x.\, \underline{2})\, \mathsf{else}\, \underline{\mathsf{div}} : (\mathsf{Nat}_\varepsilon \to \mathsf{Nat}_\varepsilon)_\bot$. We show $\underline{\mathsf{ifz}}$ can have a type of shape $(\mathsf{Nat}_\varepsilon \to \mathsf{Nat}_\varepsilon)_\bot$, meaning that it can take an input of type Nat that is guaranteed to evaluate and return a Nat that is also guaranteed to evaluate, but the overall term may diverge depending on whether the guard y is zero or not.

$$(\mathsf{IfZL2})\ \frac{\Gamma,\, y : \mathsf{Nat}_\varepsilon \vdash y : \mathsf{Nat}_\varepsilon \qquad \Gamma,\, x : \mathsf{Nat}_\varepsilon \vdash 2 : \mathsf{Nat}_\varepsilon \qquad \Gamma \vdash \underline{\mathsf{div}} : \mathsf{Nat}_\bot}{\Gamma \vdash \mathsf{ifz}\, y\, \mathsf{then}\, (\lambda x.\, \underline{2})\, \mathsf{else}\, \underline{\mathsf{div}} : (\mathsf{Nat}_\varepsilon \to \mathsf{Nat}_\varepsilon)_\bot}$$

Example 3.2. $\vdash \underline{\mathsf{div}} : \mathsf{Nat}_\bot$. We have defined $\underline{\mathsf{div}}$ as an abbreviation for $(\mathsf{fix}\, f(x).\, f\, x)\, \underline{0}$. We show $\underline{\mathsf{div}}$ has type $\mathsf{Nat}_\bot$.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash f : (\mathsf{Nat}_\varepsilon \to \mathsf{Nat}_\bot)_\varepsilon}\ \dfrac{}{\Gamma \vdash x : \mathsf{Nat}_\varepsilon}}{\Gamma,\, f : (\mathsf{Nat}_\varepsilon \to \mathsf{Nat}_\bot)_\varepsilon,\, x : \mathsf{Nat}_\varepsilon \vdash f\, x : \mathsf{Nat}_\bot}\,(\mathsf{AppR})}{\Gamma \vdash \mathsf{fix}\, f(x).\, f\, x : (\mathsf{Nat}_\varepsilon \to \mathsf{Nat}_\bot)_\varepsilon}\,(\mathsf{FixsR})\qquad \dfrac{}{\Gamma \vdash \underline{0} : \mathsf{Nat}_\varepsilon}}{\vdash (\mathsf{fix}\, f(x).\, f\, x)\, \underline{0} : \mathsf{Nat}_\bot}\,(\mathsf{AppR})$$

The tree above allows us to conclude that expressions with the term $\underline{\mathsf{div}}$ can diverge with type $\mathsf{Nat}_\bot$, so the term either evaluates to a Nat, or diverges.

Example 3.3. $\vdash \lambda x.\, \underline{\mathsf{div}} : (\mathsf{Nat}_\varepsilon \to \mathsf{Nat}_\bot)_\varepsilon$. We show $\lambda x.\, \underline{\mathsf{div}}$ evaluates, but applying it to a term will diverge.

$$(\mathsf{FixsR})\ \frac{\dfrac{}{\Gamma, x : \mathsf{Nat}_\varepsilon \vdash \underline{\mathsf{div}} : \mathsf{Nat}_\bot}}{\Gamma \vdash \lambda x.\, \underline{\mathsf{div}} : (\mathsf{Nat}_\varepsilon \to \mathsf{Nat}_\bot)_\varepsilon}$$

We know this term is guaranteed to evaluate since it is an abstraction and already a value (in normal form). However, it is not immediately possible for the expression to diverge without providing an argument, so we can annotate the outermost effect with ε corresponding to the term itself having no effect.

Example 3.4. $\vdash (\lambda y.\, \underline{2})\, \underline{\mathsf{div}} : \mathsf{Nat}_\bot$. We show $(\lambda y.\, \underline{2})\, \underline{\mathsf{div}}$ does not evaluate in CBV semantics because it will infinitely reduce the outer $\underline{\mathsf{div}}$ term before returning $\underline{2}$.

$$\dfrac{\dfrac{\dfrac{}{\Gamma, y : \mathsf{Nat}_\bot \vdash \underline{2} : \mathsf{Nat}_\varepsilon}\,(\mathsf{FixsR})}{\Gamma \vdash \lambda y.\, \underline{2} : (\mathsf{Nat}_\bot \to \mathsf{Nat}_\varepsilon)_\varepsilon}\qquad \dfrac{}{\Gamma \vdash \underline{\mathsf{div}} : \mathsf{Nat}_\bot}}{\Gamma \vdash (\lambda y.\, \underline{2})\, \underline{\mathsf{div}} : \mathsf{Nat}_\bot}\,(\mathsf{AppR})$$

Chapter 4

Soundness

To confirm the correctness of our rules, we prove type soundness, which guarantees that well-typed programs do not go wrong.

4.1 Type soundness

Type soundness (or safety) is the property of knowing that if term M is well-typed, then it does not go wrong (reduce to a stuck state). This is usually shown in two steps, known as progress and preservation:

1. (Progress) If $\vdash M : S_\alpha$, then either M is a value, or there exists N such that $M \triangleright N$.
2. (Preservation) If $\vdash M : S_\alpha$ and $M \triangleright N$ then $\vdash N : S_\alpha$

Preservation says that steps of evaluation preserve typing. Progress ensures closed, well-typed terms are either values themselves or can make a step. Together, these two theorems suffice to prove type soundness [5]. To prove progress, we initially introduced another lemma that gave us the *canonical forms* A.1 for the types defined by our grammar (i.e. the well-typed values of our types) [6]. However, we found that we could not complete a direct proof for the application case with this additional lemma. We show this fully in Section 4.2.1, but introduce the main points here for context.

The particular issue was in the application case when the overall effect of the composition was $\alpha \oplus \beta \oplus \gamma = \varepsilon$, where each metavariable was ε. This required showing the evaluation of an application MN. We know from our induction hypothesis that M and N both evaluate to some values V and W and by canonical forms, V must have had shape $(\lambda x.\, M')$. Hence, we arrive at $(\lambda x.\, M')W \triangleright M'[W/x]$, but we do not know if this final substitution evaluates. We address this issue with the use of logical relations in Section 4.2.

4.1.1 Progress

For our proof of progress theorem, we will take into account that terms and variables can exist on the left of the turnstile.

On the right of the typing judgement $\vdash M : S_\alpha$, the definition from the two-sided paper [1] specified terms M can either step or are already values. We generalise this so terms can either diverge or evaluate to a value when $\alpha = \bot$ or simply evaluate when $\alpha = \varepsilon$. The issue with the original definition was with trying to prove the preservation of our type effects in the forward direction for reduction. M being able to step with type $S_\bot$ is is already in our assumptions since $M \triangleright N$. This means we cannot conclude that N is of type $S_\bot$ without altering the definition. We explain this further in our predicate definitions Section 4.5.1

On the left of the typing judgement $M : S_\alpha \vdash$, the original definition specified that terms M can either make a step, already be a value (that is not S_α), or be stuck. For consistency with the alteration made to the meaning on the right of judgement, when $\alpha = \varepsilon$, we change this to M can either diverge, evaluate to a value (whose type is not S), or crash.

Effect Progress The progress property rules out intuitively incorrect expressions that do not reduce or are a value themselves [7]. For example, the ill-typed expression $\vdash \mathsf{ifz}\,(\lambda x.x)\,\mathsf{then}\,\underline{0}\,\mathsf{else}\,\underline{1}$ cannot make a step since the subject $(\lambda x.x)$ is a value but the whole expression is not a value and cannot make a

step. The statement and proof of progress is modified to account for effects. The approach we take to modelling effects is to provide inductive definitions of the typing judgements stating that a term M evaluates to a type S with an effect α.

Definition 4.1 (Pairs)**.** We define the first and second projections of a pairing (P, Q) by the following.

$$M \quad ::= \quad (P, Q) \mid \underline{\mathsf{fst}}\, M \ \triangleright\ P \mid \underline{\mathsf{snd}}\, M \ \triangleright\ Q$$

The following rules specify how pairs and projections behave under evaluation and can help us in proving certain properties for pairs later on.

$$(\mathsf{Proj1})\ \frac{P \ \triangleright\ Q}{\underline{\mathsf{fst}}\, P \ \triangleright\ \underline{\mathsf{fst}}\, Q} \qquad (\mathsf{Proj2})\ \frac{P \ \triangleright\ Q}{\underline{\mathsf{snd}}\, P \ \triangleright\ \underline{\mathsf{snd}}\, Q}$$

$$(\mathsf{Pair1})\ \frac{P \ \triangleright\ Q}{(P, M) \ \triangleright\ (Q, M)} \qquad (\mathsf{Pair2})\ \frac{M \ \triangleright\ N}{(V, M) \ \triangleright\ (V, N)}\, V \in val$$

4.2 Logical Relations

Logical relations are a proof method by which we can specify certain properties about programs in a language [8]. They can be used as a different method as opposed to proving these properties directly. These relations are defined over terms indexed by their type and are proven by induction on types that the property P holds for all closed terms of some type S_α.

4.2.1 Evaluation

Suppose we want to prove the case: if $\vdash M : S_\varepsilon$, then $M \Downarrow$. We introduce logical relations to help us show this case. This is a property analogous to strong normalisation, where all evaluations in the language are reduced to a normal form. It is not possible to show this case directly for an application. To demonstrate this, suppose we have a term M of shape NQ.

$$(\mathsf{AppR})\ \frac{\Gamma \vdash N : (T_\delta \to S_\alpha)_\beta, \qquad \Gamma \vdash Q : T_\delta,}{\Gamma \vdash NQ : S_{\sigma = \alpha \oplus \beta \oplus \delta},}$$

Let $\alpha, \beta, \delta = \varepsilon$ and $\sigma = \varepsilon$. We want to show $\vdash NQ : S_\varepsilon$. By the induction hypothesis, we have $N \Downarrow V$ and $Q \Downarrow W$. From the type of N we know that $N \Downarrow \lambda x.\, N'$ so it must be $V = \lambda x.\, N'$. We need to show $(\lambda x.\, N')W \Downarrow$. NQ takes the steps

$$\begin{aligned} NQ &\ \triangleright^*\ (\lambda x.\, N') \\ &\ \triangleright^*\ (\lambda x.\, N')W \\ &\ \triangleright\ N'[W/x] \quad \text{Does } N' \Downarrow? \end{aligned}$$

We know nothing about the nature of N' (it could possibly diverge). Our induction hypothesis is not strong enough to able to complete the proof. We cannot fully show the argument to the lambda abstraction has the evaluation property we are interested in.

We extend logical relations to show our desired properties for any effect case α. For our purposes, this will be whenever we know $\alpha = \varepsilon$ and M evaluates, or $\alpha = \bot$ and M diverges or evaluates to a value.

4.3 Effect Properties

To prove our effect properties, we adopt the proof structure shown in [8] and [9]. We introduce logical relations to prove the properties of our programs annotated with effects. For logical predicate $P_{S\alpha}$ and term M, the term M is accepted by the predicate if it satisfies the following conditions:

1. $\vdash M : S_\alpha$ – the term M is well-typed.
2. The property we wish to prove, $(\alpha = \bot \wedge (M \Downarrow \vee M \Uparrow)) \vee (\alpha = \varepsilon \wedge M \Downarrow V)$ i.e. $M \ \triangleright^*\ V$ where V is some value.

3. The condition is preserved by elimination forms.

Definition 4.2 (Logical Predicate)**.** Defining logical predicate $P_{S\alpha}$ over the structure of our types:

$$
\begin{aligned}
P_{\mathsf{Nat}_\alpha}(M) &::= \vdash M : \mathsf{Nat}_\alpha \wedge (\alpha = \bot \wedge (M \Downarrow \vee M \Uparrow)) \vee (\alpha = \varepsilon \wedge M \Downarrow) \\
P_{\mathsf{Ok}_\alpha}(M) &::= \vdash M : \mathsf{Ok}_\alpha \wedge (\alpha = \bot \wedge (M \Downarrow \vee M \Uparrow)) \vee (\alpha = \varepsilon \wedge M \Downarrow) \\
P_{(A \to B)_\alpha}(M) &::= \vdash M : (A \to B)_\alpha \wedge ((\alpha = \bot \wedge (M \Downarrow \lambda x.\, N \vee M \Uparrow)) \vee (\alpha = \varepsilon \wedge M \Downarrow \lambda x.\, N)) \text{ and,} \\
&\quad (M \Downarrow \lambda x.\, N \implies (\forall N'.\, P_A(N') \implies P_B((\lambda x.\, N)N'))) \\
P_{(A \rightarrowtail B)_\alpha}(M) &::= \vdash M : (A \rightarrowtail B)_\alpha \wedge (\alpha = \bot \wedge (M \Downarrow \lambda x.\, N \vee M \Uparrow)) \vee (\alpha = \varepsilon \wedge M \Downarrow \lambda x.\, N)) \text{ and,} \\
&\quad (M \Downarrow \lambda x.\, N \implies (\forall N'.\, P_B((\lambda x.\, M)N') \implies P_A(N'))) \\
P_{(A \times B)_\alpha}(M) &::= \vdash M : (A \times B)_\alpha \wedge ((\alpha = \bot \wedge (M \Downarrow (V, W) \vee M \Uparrow)) \vee (\alpha = \varepsilon \wedge M \Downarrow (V, W)) \text{ and,} \\
&\quad (M \Downarrow (V, W) \implies P_A(V) \wedge P_B(W))
\end{aligned}
$$

Definition 4.3 (Evaluation)**.** M evaluates to a value V just if closed term $M \rhd^* V$.

$$
\begin{aligned}
M \Downarrow &::= \exists V.\, M \Downarrow V \\
M \Downarrow V &::= M \rhd^* V, \quad \text{where } V \text{ is a value}
\end{aligned}
$$

Definition 4.4 (Closing Substitution)**.** We define closing substitutions γ that satisfy typing environments Γ.

$$\gamma \models \Gamma ::= \mathrm{dom}(\gamma) = \mathrm{dom}(\Gamma) \wedge \forall x \in \mathrm{dom}(\Gamma).\, P_{\Gamma(x)}(\gamma(x)).$$

Let $\gamma = \{x_1 \mapsto v_1 \ldots x_n \mapsto v_n\}$ which maps variables to values. We write $\gamma(M) = M\{v_1/x_1\}\ldots\{v_n/x_n\}$ to be the substitution in M of all variables in the domain of γ with their corresponding value. The empty substitution simply returns the term unchanged, $\emptyset(M) = M$. A closing substitution with a mapping from x to v applied to the term M is the same as applying the original closing substitution γ to the term M where x is substituted with v, i.e $\gamma[x \mapsto v](M) = \gamma(M[v/x])$.

We extend the substitution over terms. Since we allow typing of terms on the left for cases such as (AbnR), we also extend our environment to possibly contain terms and update our definition of γ. We also redefine the typing of terms to be unique to that term.

Definition 4.5 (Closing Substitutions over terms)**.**

$$\gamma \models \Gamma ::= \mathrm{dom}(\gamma) = \mathrm{dom}(\Gamma) \wedge \forall M : A \in \Gamma.\, P_A(\gamma(M))$$

Substitutions over terms are defined as follows:

$$
\begin{aligned}
\gamma(x) &::= \gamma(x) \\
\gamma(c) &::= c \\
\gamma(MN) &::= \gamma(M)\gamma(N) \\
\gamma(\lambda x.\, M) &::= \lambda x.\, \gamma(M),\ x \notin \mathrm{dom}(\gamma) \\
\gamma(\mathsf{ifz}\, M \,\mathsf{then}\, N \,\mathsf{else}\, P) &::= \mathsf{ifz}\, \gamma(M) \,\mathsf{then}\, \gamma(N) \,\mathsf{else}\, \gamma(P)
\end{aligned}
$$

In each case we are pushing the substitution in further into the term.

Lemma 4.1 (Divergence)**.** $\forall S$. *If* $\vdash M : S_\bot$ *and* $M \Uparrow$ *then* $P_{S_\bot}(M)$.

$M \Uparrow$ just if M has an infinite reduction sequence with no normal form.

Lemma 4.2 (Crashing)**.** $P_{S_\bot}(M) \implies M$ does not crash.

Lemma 4.3 (Substitution on the right)**.** *If* $\Gamma, x : B \vdash M : A$ *and* $\Gamma \vdash N : B$ *then* $\Gamma \vdash M[N/x] : A$.

Theorem E.11 reproduced from the original two-sided paper by induction on M.

Lemma 4.4 (Substitution)**.** *If* $\Gamma \vdash M : S_\alpha \wedge \gamma \models \Gamma \implies \vdash \gamma(M) : S_\alpha$

Proof. By induction on the size of γ using the substitution on the right lemma 4.3. □

4.4 Property Preservation

Lemma 4.5 (Effect Properties are preserved by forward/backward reduction)**.**

$$\text{(Forward)} \ \vdash M : S_\alpha \wedge M \ \rhd \ N \wedge P_{S\alpha}(M) \implies P_{S\alpha}(N)$$
$$\text{(Backward)} \ \vdash M : S_\alpha \wedge M \ \rhd \ N \wedge P_{S\alpha}(N) \implies P_{S\alpha}(M)$$

The proof is in two parts. We first show this for the backward part.

Part I.

Proof. By induction on S_α. In each case, assume

$$\text{(IH – Backward)} \ \vdash M : S_\alpha \wedge M \ \rhd \ N \wedge P_{S\alpha}(N) \tag{4.1}$$

and show $P_{S\alpha}(M)$ for each effect α.

Case($\mathbf{Nat}_\alpha$).

1. $\alpha = \varepsilon$: Suppose $P_{\mathsf{Nat}_\varepsilon}(N)$.
   We need to show $P_{\mathsf{Nat}_\varepsilon}(M)$ where $M \ \rhd \ N$.
   - We have that $N \Downarrow$.
   - Then it must be that $M \Downarrow$.
   - And we can conclude $P_{\mathsf{Nat}_\varepsilon}(M)$.
2. $\alpha = \bot$: Suppose $P_{\mathsf{Nat}_\bot}(N)$, then $N \Uparrow \vee N \Downarrow$.
   We need to show $P_{\mathsf{Nat}_\bot}(M)$ for each case on N above.
   - If $P_{\mathsf{Nat}_\bot}(N)$, then $N \Uparrow$ or $N \Downarrow$
   - In the case $N \Uparrow$ then $M \Uparrow$ since $M \ \rhd \ N$.
   - In the case $N \Downarrow$ then $M \Downarrow V$ for some value V.
   - And we can conclude $P_{\mathsf{Nat}_\bot}(M)$.

Case($\mathbf{Ok}_\alpha$). Similar to **Case($\mathbf{Nat}_\alpha$).**

Case($\mathbf{A \to B}$)$_\alpha$.

1. $\alpha = \varepsilon$: Suppose $P_{(A\to B)_\varepsilon}(N)$. Then $N \Downarrow \lambda x.\, N' \implies (\forall Q. P_A(Q) \implies P_B((\lambda x.\, N')Q))$.
   We need to show $P_{(A\to B)_\varepsilon}(M)$ where $M \ \rhd \ N$.
   - Suppose $M \Downarrow \lambda x.\, M'$.
   - We need to show $\forall Q.\, P_A(Q) \implies P_B((\lambda x.\, M')Q)$.
   - But since $\rhd$ is deterministic, we know M and N evaluate to the same value.
   - Hence $N \Downarrow \lambda x.\, M'$, and $\forall Q.\, P_A(Q) \implies P_B((\lambda x.\, M')Q)$ which is precisely what we wanted to show.
2. $\alpha = \bot$: Suppose $P_{(A\to B)_\bot}(N)$, then $N \Uparrow$ or $N \Downarrow \lambda x.\, N' \implies (\forall Q.\, P_A(Q) \implies P_B((\lambda x.\, N')Q))$.
   We need to show the properties of $P_{(A\to B)_\bot}(M)$ for each case on N above.
   - In the case $N \Uparrow$ we know $M \Uparrow$ since $M \ \rhd \ N$, hence $P_{(A\to B)_\bot}(M)$ by our divergence lemma.
   - In the case $N \Downarrow \lambda x.\, N'$, we know $M \Downarrow \lambda x.\, M'$ and that $N' = M'$ by the same reasoning as above in 1.

Case($\mathbf{A \rightarrowtail B})_\alpha$.

1. $\alpha = \varepsilon$: Suppose $P_{(A \rightarrowtail B)_\varepsilon}(N)$, then $N \Downarrow \lambda x.\, N' \implies (\forall Q.\, P_B((\lambda x.\, N')Q) \implies P_A(Q))$.
   We need to show $P_{(A \rightarrowtail B)_\varepsilon}(M)$, where $M \rhd N$.
   - Suppose $M \Downarrow \lambda x.\, M'$.
   - We need to show $\forall Q. P_B((\lambda x.\, M')Q) \implies P_A(Q))$.
   - Since $\rhd$ is deterministic, we know M and N evaluate to the same value.
   - Hence $N \Downarrow \lambda x.\, M'$ and we get $\forall Q. P_B((\lambda x.\, M')Q \implies P_A(Q))$ as required.
2. $\alpha = \bot$: Suppose $P_{(A \rightarrowtail B)_\bot}(N)$, we have $N \Uparrow$ or $N \Downarrow \lambda x.\, N' \implies (\forall Q.\, P_B((\lambda x.\, N')Q) \implies P_A(Q))$.
   We need to show the properties of $P_{(A \rightarrowtail B)_\bot}(M)$ for each case on N above.
   - In the case $N \Uparrow$, we know $M \Uparrow$ since $M \rhd N$, so $P_{(A \rightarrowtail B)_\bot}(M)$ by the divergence lemma.
   - In the case $N \Downarrow \lambda x.\, N'$, we know $M \Downarrow \lambda x.\, M'$ and that $N' = M'$ by the same reasoning as above in 1.

Case($\mathbf{A \times B})_\alpha$.

1. $\alpha = \varepsilon$: Suppose $P_{(A \times B)_\varepsilon}(N)$, then $N \Downarrow (V, W) \implies (P_A(V)$ and $P_B(W))$.
   Need to show: $P_{(A \times B)_\varepsilon}(M)$ where $M \rhd N$.
   - We have $P_{(A \times B)_\varepsilon}(N)$.
   - Suppose $M \Downarrow (T, U)$ for some T and U values.
   - We need to show $P_A(T) \wedge P_B(U)$.
   - Since $M \rhd N$, it must be that $V = T$ and $W = U$.
   - From this we have $N \Downarrow (T, U)$ giving $P_A(T)$ and $P_B(U)$ which is what we wanted to show.
2. $\alpha = \bot$: Suppose $P_{(A \times B)_\bot}(N)$. Then $N \Uparrow$ or $N \Downarrow (V, W) \implies (P_A(V)$ and $P_B(W))$.
   Need to show: $P_{(A \times B)_\bot}(M)$ where $M \rhd N$, and $M \Uparrow$ or $M \Downarrow (T, U) \implies (P_A(T) \wedge P_B(U))$.
   - From the definition of $P_{(A \times B)_\bot}(M)$, we have $M \Uparrow$ or $M \Downarrow (V, W)$.
   - In the case $N \Uparrow$, we know $M \Uparrow$ and hence $P_{(A \times B)_\bot}(M)$ by the divergence lemma.
   - In the case $M \Downarrow$ the proof follows as above in 1.

□

We now show this for the forward part.

Part II.

Proof. By induction on S_α.

$$(\text{IH} - \text{Forward}) \ \vdash M : S_\alpha \wedge M \rhd N \wedge P_{S\alpha}(M) \tag{4.2}$$

and show $P_{S\alpha}(N)$ for each effect α.

Case($\mathbf{Nat}_\alpha$).

1. $\alpha = \varepsilon$: Suppose $P_{\mathsf{Nat}_\varepsilon}(M)$.
   Need to show $P_{\mathsf{Nat}_\varepsilon}(N)$ where $M \rhd N$.
   - We have $M \Downarrow$.
   - Then it must be that $N \Downarrow$.
   - From this we can conclude $P_{\mathsf{Nat}_\varepsilon}(N)$.
2. $\alpha = \bot$: Suppose $P_{\mathsf{Nat}_\bot}(M)$. Then we have $M \Uparrow \vee M \Downarrow$.
   We need to show $P_{\mathsf{Nat}_\bot}(N)$ in the case $M \Uparrow$ or $M \Downarrow$.
   - If $P_{\mathsf{Nat}_\bot}(M)$ then $M \Uparrow$ or $M \Downarrow$
   - In the case $M \Uparrow$ then $N \Uparrow$ by confluence, since $M \rhd N$.
   - In the case $M \Downarrow$ then M reduces to some value and it must be that $N \Downarrow$.
   - From this we can conclude $P_{\mathsf{Nat}_\bot}(N)$.

Case($\mathbf{Ok}_\alpha$). Similar to **Case($\mathbf{Nat}_\alpha$).**

Case($\mathbf{A}\to\mathbf{B}$)$_\alpha$.

1. $\alpha = \varepsilon$: Suppose $P_{(A\to B)_\varepsilon}(M)$ then $M \Downarrow \lambda x.\, M'$.
   We need to show $P_{(A\to B)_\varepsilon}(N)$ where $M \rhd N$.
   - Suppose $N \Downarrow \lambda x.\, N'$. Need to show $\forall Q.\, P_A(Q) \implies P_B((\lambda x.\, N')Q)$.
   - Since $\rhd$ is deterministic, we know $M' = N'$.
   - Hence $M \Downarrow \lambda x.\, N'$ and $\forall Q.\, P_A(Q) \implies P_B((\lambda x.\, N')Q)$ which is exactly what we wanted to show.
2. $\alpha = \bot$: Suppose $P_{(A\to B)_\bot}(M)$, then $M \Uparrow$ or $M \Downarrow \lambda x.\, M' \implies \forall Q.\, P_A(Q) \implies P_B((\lambda x.\, M')Q)$.
   We need to show the properties of $P_{(A\to B)_\bot}(N)$ for each case on M above.
   - In the case $M \Uparrow$ we know $N \Uparrow$. This gives $P_{(A\to B)_\bot}(N)$ by the divergence lemma.
   - In the case $M \Downarrow \lambda x.\, M'$, we know $N \Downarrow \lambda x.\, N'$ and that $N' = M'$ by the same reasoning as above in 1.

Case($\mathbf{A}\rightarrowtail\mathbf{B}$)$_\alpha$.

1. $\alpha = \varepsilon$: Suppose $P_{(A\rightarrowtail B)_\varepsilon}(M)$ then $M \Downarrow \lambda x.\, M' \implies (\forall Q.\, P_B((\lambda x.\, M')Q) \implies P_A(Q))$.
   We need to show $P_{(A\rightarrowtail B)_\varepsilon}(N)$, where $M \rhd N$.
   - Suppose $N \Downarrow \lambda x.\, N'$. We need to show $\forall Q. P_B((\lambda x.\, N')Q) \implies P_A(Q))$.
   - Since $\rhd$ is deterministic, we know M and N evaluate to the same value.
   - Hence $M \Downarrow \lambda x.\, N'$ and we get $\forall Q. P_B((\lambda x.\, N')Q \implies P_A(Q))$ as required.
2. $\alpha = \bot$: Suppose $P_{(A\rightarrowtail B)_\bot}(N)$, we have $N \Uparrow$ or $N \Downarrow \lambda x.\, N' \implies (\forall Q.\, P_B((\lambda x.\, N')Q) \implies P_A(N))$.
   We need to show the properties of $P_{(A\rightarrowtail B)_\bot}(N)$ for each case on M above.
   - In the case $M \Uparrow$, we know $N \Uparrow$ since $M \rhd N$. Hence $P_{(A\rightarrowtail B)_\bot}(N)$ by divergence lemma.
   - In the case $M \Downarrow \lambda x.\, M'$, we know $M \Downarrow \lambda x.\, N'$ and that $N' = M'$ by the same reasoning as above in 1.

Case($\mathbf{A}\times\mathbf{B}$)$_\alpha$.

1. $\alpha = \varepsilon$: Suppose $P_{(A\times B)_\varepsilon}(M)$. Then $M \Downarrow (V, W)$ for some pair of values V and W.
   Need to show: $P_{(A\times B)_\varepsilon}(N)$ where $M \rhd N$.
   - We have $P_{(A\times B)_\varepsilon}(M)$ so we know $P_A(V)$ and $P_B(W)$.
   - To show $P_{(A\times B)_\varepsilon}(N)$, first assume $N \Downarrow (T, U)$ for some T and U and show $P_A(T)$ and $P_B(U)$.
   - Since $M \rhd N$ and $\rhd$ is deterministic, it must be that $V \Downarrow T$ and $W \Downarrow U$, i.e. the pair evaluates to the same values.
   - Hence we have $M \Downarrow (T, U)$ which gives us $P_A(T) \wedge P_B(U)$ as required.
2. $\alpha = \bot$: Suppose $P_{(A\times B)_\bot}(M)$.
   Need to show: $P_{(A\times B)_\bot}(N)$ where $M \rhd N$.
   - From the definition of $P_{(A\times B)_\bot}(M)$, we have $M \Uparrow$ or $M \Downarrow (V, W)$.
   - In the case $M \Uparrow$, we know $N \Uparrow$ and $P_{(A\times B)_\bot}(N)$ by the divergence lemma.
   - In the case $M \Downarrow$, the proof follows like above in 1.

□

Using preservation, we can now prove the effect progress theorem. We want to show every term of type S_α is in P_{S_α}. Since the proof will be by induction over the typing derivations, we need to consider open terms (in the abstraction case). So, we generalise the induction hypothesis to be over all closed instances of an open term M using our definition for closing substitutions 4.4 [8, 9].

4.5 Effect Progress

Theorem 4.5.1 (Effect Progress). $\Gamma \vdash M : S_\alpha \wedge \gamma \models \Gamma \implies P_{S_\alpha}(\gamma(M))$

Proof. By induction on $\Gamma \vdash M : S_\alpha$.

Case(Id). In this case M is of shape x. Let $A = S_\alpha$.

$$(\mathsf{Id})\ \frac{}{\Gamma,\, x : A \vdash x : A,}$$

Need to show $P_A(\gamma(x))$. We have $\gamma \models \Gamma$. Case follows by definition of $\Gamma \models \gamma$, and we have $P_{\Gamma(x)}(\gamma(x))$. $\Gamma(x) = A$, therefore $P_A(\gamma(x))$ as required.

Case(ZeroR). In this case M is of shape zero.

$$(\mathsf{ZeroR})\ \frac{}{\Gamma \vdash \mathsf{zero} : \mathsf{Nat}_\varepsilon,}$$

Need to show $P_{\mathsf{Nat}_\varepsilon}(\gamma(\mathsf{zero}))$. We have $\gamma \models \Gamma$. Since $\mathsf{zero} = \gamma(\mathsf{zero})$, our goal becomes $P_{\mathsf{Nat}_\varepsilon}(\mathsf{zero})$. From the our predicate definition, it suffices to show $\alpha = \varepsilon \wedge M \Downarrow$. We know $\mathsf{zero} \Downarrow$ as it is a numeric value.

Case(SuccR). In this case M is of shape $\mathsf{succ}(N)$.

$$(\mathsf{cR})\ \frac{\Gamma \vdash N : \mathsf{Nat}_\alpha,}{\Gamma \vdash \mathsf{succ}(N) : \mathsf{Nat}_\alpha,}$$

Need to show $P_{\mathsf{Nat}_\alpha}(\gamma(\mathsf{succ}(N)) \equiv P_{\mathsf{Nat}_\alpha}(\mathsf{succ}(\gamma(N))$. We have $\gamma \models \Gamma$. By the induction hypothesis, we have $P_{\mathsf{Nat}_\alpha}(\gamma(N))$ and from the definition of the first property, either:

1. $\alpha = \bot \wedge\ (\gamma(N) \Downarrow \vee \gamma(N) \Uparrow)$, or
2. $\alpha = \varepsilon \wedge \gamma(N) \Downarrow$

   Cases of α:

   - $\alpha = \varepsilon$

     We have $\gamma(N) \Downarrow$. Need to show $\mathsf{succ}(\gamma(N)) \Downarrow$.

     – We know $\mathsf{succ}(\gamma(N)) \rhd^* \mathsf{succ}(V)$ where V is some value, by CBV evaluation rules. We also know $P_{\mathsf{Nat}_\varepsilon}(\mathsf{succ}(V))$ – the successor of a numeral evaluates, by forward preservation. Hence $P_{\mathsf{Nat}_\varepsilon}(\mathsf{succ}(\gamma(N)))$ by backward preservation.

   - $\alpha = \bot$

     We have $\gamma(N) \Downarrow \vee \gamma(N) \Uparrow$. Need to show $\mathsf{succ}(\gamma(N)) \Downarrow \vee\ \mathsf{succ}(\gamma(N)) \Uparrow$. We show this for the second part.

     – Since $N \Uparrow$, we know it has an infinite reduction sequence. Repeatedly performing steps from N means succ will never be evaluated in CBV. Hence $\mathsf{succ}(N) \Uparrow$ and $P_{\mathsf{Nat}_\bot}(\mathsf{succ}(\gamma(N)))$ by the divergence lemma 4.1.

Case(PredR). Similar to **(SuccR).**

Case(AbsR). In this case M is of shape $\lambda x.\, N$.
For simplicity, we consider $f \notin FV(N)$, hence the following rule:

$$(\mathsf{AbsR})\ \frac{\Gamma,\, x : T_\delta \vdash N : S_\alpha,}{\Gamma \vdash \lambda x.\, N : (T_\delta \to S_\alpha)_\varepsilon,}$$

Need to show $P_{(T_\delta \to S_\alpha)_\varepsilon}(\gamma(\lambda x.\, N)) \equiv P_{(T_\delta \to S_\alpha)_\varepsilon}(\lambda x.\, \gamma(N))$. We have $\gamma \models \Gamma$. Considering each case for α and δ, we have to show three possible subcases:

1. $(\delta = \varepsilon,\, \alpha = \varepsilon)$. Show $P_{(T_\varepsilon \to S_\varepsilon)_\varepsilon}(\lambda x.\, \gamma(N))$ – the abstraction evaluates

2. $(\delta = \bot, \alpha = \bot)$. Show $P_{(T_\bot \to S_\bot)_\varepsilon}(\lambda x.\, \gamma(N))$ – the input and body of the abstraction can diverge, but the abstraction evaluates
3. $(\delta = \varepsilon, \alpha = \bot)$. Show $P_{(T_\varepsilon \to S_\bot)_\varepsilon}(\lambda x.\, \gamma(N))$ – the body of the abstraction can diverge, but the abstraction evaluates

Our induction hypotheses read:

(IH) $\Gamma, x : T_\delta \vdash N : S_\alpha, \wedge\gamma' \models \Gamma, x : T_\delta \implies P_{S_\alpha}(\gamma'(N))$

IH 1. $\Gamma, x : T_\varepsilon \vdash N : S_\varepsilon, \wedge\gamma' \models \Gamma, x : T_\varepsilon \implies P_{S_\varepsilon}(\gamma'(N))$
IH 2. $\Gamma, x : T_\bot \vdash N : S_\bot, \wedge\gamma' \models \Gamma, x : T_\bot \implies P_{S_\bot}(\gamma'(N))$
IH 3. $\Gamma, x : T_\varepsilon \vdash N : S_\bot, \wedge\gamma' \models \Gamma, x : T_\varepsilon \implies P_{S_\bot}(\gamma'(N))$

Subcase 1. $(\delta = \varepsilon, \alpha = \varepsilon)$ Suffices to show the properties of $P_{(T_\delta \to S_\alpha)_\varepsilon}$

(1) $\lambda x.\, \gamma(N) \Downarrow$ – abstractions are values

(2) $\forall N'.P_{T_\varepsilon}(N') \implies P_{S_\varepsilon}((\lambda x.\, \gamma(N))N')$

Since M is already an abstraction, the condition $M \Downarrow \lambda x.\, M'$ is satisfied. Assume $P_{T_\varepsilon}(N')$. To prove the third property (3) we need to show $P_{S_\varepsilon}((\lambda x.\, \gamma(N))N')$. From our assumption we have $N' \Downarrow V$ for some value V. Using forward ($\implies$) part of the effect preservation lemma 4.5, V satisfies $P_{T_\varepsilon}(V)$. We then substitute V for x in the function in our extended environment. By the induction hypothesis (IH1), with $\gamma[V/x] \models \Gamma, x : T_\varepsilon$, we get $P_{S_\varepsilon}(\gamma[V/x](N))$. Consider the evaluation:

$$(\lambda x.\, \gamma(N))N' \rhd^* (\lambda x.\, \gamma(N))V \tag{4.3}$$
$$\rhd \gamma(N)[V/x] \equiv \gamma[V/x](N) \tag{4.4}$$

We use $P_{S_\varepsilon}(\gamma[V/x](N))$ with the fact $(\lambda x.\, \gamma(N))N' \rhd^* \gamma[V/x](N)$ and the backward ($\impliedby$) part of our preservation lemma 4.5 which gives us $P_{S_\varepsilon}((\lambda x.\, \gamma(N))N')$ as required.

Subcase 2. $(\delta = \bot, \alpha = \bot)$ Suffices to show the properties of $P_{(T_\delta \to S_\alpha)_\varepsilon}$

(1) $\vdash \lambda x.\, \gamma(N) : (T_\bot \to S_\bot)_\varepsilon$ – by substitution lemma 4.4

(2) $\lambda x.\, \gamma(N) \Downarrow$ – abstractions are values

(3) $\forall N'.P_{T_\bot}(N') \implies P_{S_\bot}((\lambda x.\, \gamma(N))N')$

Assume $P_{T_\bot}(N')$. To prove the second property (2), we need to show $P_{S_\bot}((\lambda x.\, \gamma(N))N')$. That is, $(\lambda x.\, \gamma(N)))N'$ either evaluates or diverges in the case N' evaluates or diverges from our assumption.

- In the case $N' \Uparrow$, we know that the application never evaluates in CBV since the outer term N' is continuously reduced in an infinite sequence. So it must be that $(\lambda x.\, \gamma(N))N' \Uparrow$.
- From our assumption we know $\vdash N' : T_\bot$, and with (1), we have $\vdash (\lambda x.\, \gamma(N))N' : S_\bot$ by (AppR).
- This gives $P_{S_\bot}((\lambda x.\, \gamma(N))N')$ by our divergence lemma 4.1.
- In the case $N' \Downarrow V$ for some value V, we can instantiate $\gamma' = \gamma[x \mapsto V]$.

By the induction hypothesis (IH2), with $\gamma[V/x] \models \Gamma, x : T_\bot$ we have $P_{S_\bot}(\gamma[V/x](N))$. Use $P_{S_\bot}(\gamma[V/x](N))$ with the fact $(\lambda x.\, \gamma(N))N' \rhd^* \gamma[V/x](N)$ and the backward ($\impliedby$) part of our preservation lemma 4.5, which gives us $P_{S_\bot}((\lambda x.\, \gamma(N))N')$ as required.

Subcase 3. $(\delta = \varepsilon, \alpha = \bot)$ Suffices to show the properties of $P_{(T_\delta \to S_\alpha)_\varepsilon}$

(1) $\vdash \lambda x.\, \gamma(N) : (T_\varepsilon \to S_\bot)_\varepsilon$ – by substitution lemma 4.4

(2) $\lambda x.\, \gamma(N) \Downarrow$ – abstractions are values

(3) $\forall N'.P_{T_\varepsilon}(N') \implies P_{S_\bot}((\lambda x.\, \gamma(N))N')$

Assume $P_{T_\varepsilon}(N')$. To prove the second property (2), we need to show $P_{S_\perp}((\lambda x.\,\gamma(N))N')$. From our assumption we have $N' \Downarrow V$ for some value V. Need to show $(\lambda x.\,\gamma(N))N'$ either diverges or evaluates in the case N' evaluates. Since we know that N' evaluates, we have the following evaluation:

- $(\lambda x.\,\gamma(N))N' \rhd^* (\lambda x.\,\gamma(N))V \rhd \gamma(N)[V/x] \equiv \gamma[V/x](N)$

By the induction hypothesis (IH3), with $\gamma[V/x] \models \Gamma, x : T_\varepsilon$, we get $P_{S_\perp}(\gamma[V/x](N))$. Use $P_{S_\perp}(\gamma[V/x](N))$ with $(\lambda x.\,\gamma(N))V \rhd^* \gamma[V/x](N)$ and the backward ($\Longleftarrow$) part of our preservation lemma 4.5, which gives us $P_{S_\perp}((\lambda x.\,\gamma(N))N')$ as required.

Case(AbnR). In this case M is of shape $\lambda x.\,N$.

$$\text{(AbnR)}\ \frac{\Gamma,\ N : S_\alpha \vdash x : T_\delta,}{\Gamma \vdash \lambda x.\,N : (T_\delta \rightarrowtail S_\alpha)_\varepsilon,}$$

Need to show $P_{(T_\delta \rightarrowtail S_\alpha)_\varepsilon}(\gamma(\lambda x.\,N)) \equiv P_{(T_\delta \rightarrowtail S_\alpha)_\varepsilon}(\lambda x.\,\gamma(N))$. We have $\gamma \models \Gamma$. Considering each case for α and δ, we have to show three possible subcases:

1. $(\delta = \varepsilon,\ \alpha = \varepsilon)$. Show $P_{(T_\varepsilon \rightarrowtail S_\varepsilon)_\varepsilon}(\lambda x.\,\gamma(N))$ – the abstraction evaluates
2. $(\delta = \perp,\ \alpha = \perp)$. Show $P_{(T_\perp \rightarrowtail S_\perp)_\varepsilon}(\lambda x.\,\gamma(N))$ – the input and body of the abstraction can diverge, but the abstraction evaluates
3. $(\delta = \varepsilon,\ \alpha = \perp)$. Show $P_{(T_\varepsilon \rightarrowtail S_\perp)_\varepsilon}(\lambda x.\,\gamma(N))$ – the body of the abstraction can diverge, but the abstraction evaluates

Our induction hypotheses read:

(IH) $\Gamma,\ N : S_\alpha \vdash x : T_\delta,\ \wedge\gamma' \models \Gamma, N : S_\alpha \implies P_{T_\delta}(\gamma'(x))$

IH 1. $\Gamma,\ N : S_\varepsilon \vdash x : T_\varepsilon,\ \wedge\gamma' \models \Gamma, N : S_\varepsilon \implies P_{T_\varepsilon}(\gamma'(x))$
IH 2. $\Gamma,\ N : S_\perp \vdash x : T_\perp,\ \wedge\gamma' \models \Gamma, N : S_\perp \implies P_{T_\perp}(\gamma'(x))$
IH 3. $\Gamma,\ N : S_\perp \vdash x : T_\varepsilon,\ \wedge\gamma' \models \Gamma, N : S_\perp \implies P_{T_\varepsilon}(\gamma'(x))$

Subcase 1. $(\delta = \varepsilon,\ \alpha = \varepsilon)$ Suffices to show the properties of $P_{(T_\delta \rightarrowtail S_\alpha)_\varepsilon}$

(1) $\vdash \lambda x.\,\gamma(N) : (T_\varepsilon \rightarrowtail S_\varepsilon)_\varepsilon$ – by substitution lemma 4.4

(2) $\lambda x.\,\gamma(N) \Downarrow$ – abstractions are values

(3) $\forall N'.P_{S_\varepsilon}((\lambda x.\,\gamma(N))N') \implies P_{T_\varepsilon}(N')$

Since M is already an abstraction, the precondition for property (2) is satisfied. Assume $P_{S_\varepsilon}((\lambda x.\,\gamma(N))N')$. We need to show $P_{T_\varepsilon}(N')$. From our assumption we have $(\lambda x.\,\gamma(N))N' \Downarrow V$. It must be that $N' \Downarrow W$ for some value W. We already have $\gamma \models \Gamma$. We instantiate $\gamma' = \gamma[x \mapsto W]$. Consider the evaluation:

$$P_{S_\varepsilon}((\lambda x.\,\gamma(N))N') \rhd^* P_{S_\varepsilon}((\lambda x.\,\gamma(N))W) \text{ – by forwards preservation} \tag{4.5}$$
$$\rhd P_{S_\varepsilon}(\gamma[W/x](N)) \equiv P_{S_\varepsilon}(\gamma'(N)) \text{ – by forwards preservation} \tag{4.6}$$

With $P_{S_\varepsilon}(\gamma'(N))$ and (IH1) we have $P_{T_\varepsilon}(\gamma'(x)) \equiv P_{T_\varepsilon}(\gamma[W/x](x)) \equiv P_{T_\varepsilon}(W)$. Since $N' \Downarrow W$, backwards preservation gives us $P_{T_\varepsilon}(N')$.

Subcase 2. $(\delta = \perp,\ \alpha = \perp)$ Suffices to show the properties of $P_{(T_\delta \rightarrowtail S_\alpha)_\varepsilon}$

(1) $\vdash \lambda x.\,\gamma(N) : (T_\perp \rightarrowtail S_\perp)_\varepsilon$ – by substitution lemma 4.4

(2) $\lambda x.\,\gamma(N) \Downarrow$ – abstractions are values

(3) $\forall N'.P_{S_\perp}((\lambda x.\,\gamma(N))N') \implies P_{T_\perp}(N')$

Assume $P_{S_\perp}((\lambda x.\,\gamma(N))N')$. From our assumption we have $(\lambda x.\,\gamma(N))N'$ evaluates or diverges. We need to show $P_{T_\perp}(N')$, that is either $N' \Uparrow$ or $N' \Downarrow$.

- Consider the case $(\lambda x.\,\gamma(N))N' \Uparrow$. It must be that either $N' \Uparrow$ or $\gamma[W/x](N) \Uparrow$.

- In the case N' diverges, we know $N' \Uparrow$. We also know $\vdash (\lambda x.\, \gamma(N))N' : S_\perp$ from our assumption, so $\vdash N' : T_\perp$ by inversion.
- This gives $P_{T_\perp}(N')$ by our divergence lemma 4.1.
- In the case $\gamma[W/x](N) \Uparrow$, it must have been that $N' \Downarrow W$ for some value W for us to reach the body of the abstraction. We reach the same conclusion for when $(\lambda x.\, \gamma(N))N' \Downarrow$.

We have $\gamma \models \Gamma$. Instantiate $\gamma' = \gamma[x \mapsto W]$. Consider the evaluation in the case $N' \Downarrow W$:

$$P_{S_\perp}((\lambda x.\, \gamma(N))N') \rhd^* P_{S_\perp}((\lambda x.\, \gamma(N))W) \text{ – by forwards preservation} \tag{4.7}$$
$$\rhd P_{S_\perp}(\gamma[W/x](N)) \equiv P_{S_\perp}(\gamma'(N)) \text{ – by forwards preservation} \tag{4.8}$$

With $P_{S_\perp}(\gamma'(N))$ and (IH2) we have $P_{T_\perp}(\gamma'(x)) \equiv P_{T_\perp}(\gamma[W/x](x)) \equiv P_{T_\perp}(W)$. In the case $N' \Downarrow W$, backwards preservation gives us $P_{T_\perp}(N')$.

Subcase 3. ($\delta = \varepsilon$, $\alpha = \perp$) Suffices to show the properties of $P_{(T_\delta \rightarrowtail S_\alpha)_\varepsilon}$

(1) $\vdash \lambda x.\, \gamma(N) : (T_\varepsilon \rightarrowtail S_\perp)_\varepsilon$ – by substitution lemma 4.4

(2) $\lambda x.\, \gamma(N) \Downarrow$ – abstractions are values

(3) $\forall N'. P_{S_\perp}((\lambda x.\, \gamma(N))N') \implies P_{T_\varepsilon}(N')$

Assume $P_{S_\perp}((\lambda x.\, \gamma(N))N')$. We have $\gamma \models \Gamma$. From our assumption we have $(\lambda x.\, \gamma(N))N'$ diverges or evaluates. To prove the second property (2) we need to show $P_{T_\varepsilon}(N')$, N' evaluates.

- Consider the case $(\lambda x.\, \gamma(N))N' \Uparrow$. We know it must have been that $N' \Downarrow W$ when $\gamma[W/x](N) \Uparrow$.
- This is also true when $(\lambda x.\, \gamma(N))N' \Downarrow$ since it must have been $N' \Downarrow W$.

Instantiate $\gamma' = \gamma[x \mapsto W]$, considering the evaluation:

$$P_{S_\perp}((\lambda x.\, \gamma(N))N') \rhd^* P_{S_\perp}((\lambda x.\, \gamma(N))W) \text{ – by forwards preservation} \tag{4.9}$$
$$\rhd P_{S_\perp}(\gamma[W/x](N)) \equiv P_{S_\perp}(\gamma'(N)) \text{ – by forwards preservation} \tag{4.10}$$

With $P_{S_\perp}(\gamma'(N))$ and (IH3) we have $P_{T_\varepsilon}(\gamma'(x)) \equiv P_{T_\varepsilon}(\gamma[W/x](x)) \equiv P_{T_\varepsilon}(W)$. Since $N' \Downarrow W$, backwards preservation gives us $P_{T_\varepsilon}(N')$.

Case(AppR). In this case R is of shape MN.

$$(\mathsf{AppR})\ \frac{\Gamma \vdash M : (T_\delta \to S_\alpha)_\beta, \qquad \Gamma \vdash N : T_\delta,}{\Gamma \vdash MN : S_{\sigma = \alpha \oplus \beta \oplus \delta},}$$

Need to show $P_{S_\sigma}(\gamma(MN)) \equiv P_{S_\sigma}(\gamma(M)\gamma(N))$. We have $\gamma \models \Gamma$. By the induction hypothesis, we have

(1) $P_{(T_\delta \to S_\alpha)_\beta}(\gamma(M))$, and

(2) $P_{T_\delta}(\gamma(N))$

By definition of the 3rd property of (1) we have:

$$\gamma(M) \Downarrow \lambda x.\, M' \implies (\forall Q.\, P_{T_\delta}(Q) \implies P_{S_\alpha}((\lambda x.\, M')Q))$$

Cases of σ:

- $\sigma = \alpha \oplus \beta \oplus \delta = \varepsilon$. The application evaluates. Show $P_{S_\varepsilon}(\gamma(N)\gamma(Q))$. We know $\gamma(M)$ evaluates to an abstraction $(\lambda x.\, M')$, and have

  $$\forall Q.\, P_{T_\varepsilon}(Q) \implies P_{S_\varepsilon}((\lambda x.\, M')Q).$$

  Instantiating the above with (2) and $\delta = \varepsilon$, we get $P_{S_\varepsilon}(\gamma(M)\gamma(N))$.
- $\sigma = \alpha \oplus \beta \oplus \delta = \perp$. The application either evaluates or diverges. Show $P_{S_\perp}(\gamma(M)\gamma(N))$. We know $\gamma(M)$ evaluates to an abstraction $\lambda x.\, M'$, or diverges. Consider each case separately:
  - In the case $\gamma(M) \Downarrow \lambda x.\, M'$ we have

    $$\forall Q.\, P_{T_\perp}(Q) \implies P_{S_\perp}((\lambda x.\, M')Q).$$

    Instantiating the above with (2) and $\delta = \perp$, we get $P_{S_\perp}(\gamma(M)\gamma(N))$ as required.
  - Otherwise if $\gamma(M) \Uparrow$ we know the application diverges, $\gamma(M)\gamma(N) \Uparrow$ and hence $P_{S_\perp}(\gamma(M)\gamma(N))$ by the divergence lemma.

Case(PairR). In this case M is of shape (N, Q).

$$(\mathsf{PairR})\ \frac{\Gamma \vdash N : S_\alpha, \qquad \Gamma \vdash Q : T_\beta,}{\Gamma \vdash (N, Q) : (S_\alpha \times T_\beta)_{\alpha\oplus\beta},}$$

Need to show $P_{(S_\alpha \times T_\beta)_{\alpha\oplus\beta}}(\gamma(N,Q)) \equiv P_{(S_\alpha \times T_\beta)_{\alpha\oplus\beta}}(\gamma(N),\gamma(Q))$. We have $\gamma \models \Gamma$. By the induction hypothesis, we have

1. $P_{S_\alpha}(\gamma(N))$, and
2. $P_{T_\beta}(\gamma(Q))$

Subcase. $(\alpha = \varepsilon,\ \beta = \varepsilon)$. We have

- $P_{S_\varepsilon}(\gamma(N))$, and
- $P_{T_\varepsilon}(\gamma(Q))$

Need to show: $P_{(S_\varepsilon \times T_\varepsilon)_\varepsilon}(\gamma(N),\gamma(Q))$ and $M \Downarrow (V,W) \implies (P_A(\underline{\mathsf{fst}}\,\gamma(M)) \wedge P_B(\underline{\mathsf{snd}}\,\gamma(M))$. Suppose $M \Downarrow (V,W)$, and we show the second part of the implication.

By forward ($\Longrightarrow$) part of our preservation lemma 4.5, we know $P_{S_\varepsilon}(V)$ and $P_{T_\varepsilon}(W)$.

Know: $\underline{\mathsf{fst}}(\gamma(N),\gamma(Q)) \rhd^* V$ and $\underline{\mathsf{snd}}(\gamma(N),\gamma(Q)) \rhd^* W$, where $(V,W)\,val$.

By backward ($\Longleftarrow$) part of our preservation lemma 4.5, we get

$$P_{S_\varepsilon}(\underline{\mathsf{fst}}(\gamma(N),\gamma(Q)))$$
$$P_{T_\varepsilon}(\underline{\mathsf{snd}}(\gamma(N),\gamma(Q)))$$

By definition this is $P_{(S_\varepsilon \times T_\varepsilon)_\varepsilon}(\gamma(N),\gamma(Q))$.

Let $\sigma = \alpha \oplus \beta = \bot$. In the following cases, since the overall effect of the pair is divergence, we do not need to show the third property of $P_{(A\times B)_\sigma}$.

Subcase. $(\alpha = \bot,\ \beta = \varepsilon)$. Have

- $P_{S_\bot}(\gamma(N))$, and
- $P_{T_\varepsilon}(\gamma(Q))$ from the induction hypothesis.

Need to show: $P_{(S_\bot \times T_\varepsilon)_\bot}(\gamma(N),\gamma(Q))$, that is, $\gamma(M) \Uparrow$.

- In the case $\gamma(N) \Uparrow$, then we know the pair diverges $(\gamma(N),\gamma(Q)) \Uparrow$, and hence $P_{(S_\bot \times T_\varepsilon)_\bot}(\gamma(N),\gamma(Q))$ by our divergence lemma 4.1.

Subcase. $(\alpha = \varepsilon,\ \beta = \bot)$. We have

- $P_{S_\varepsilon}(\gamma(N))$, and
- $P_{T_\bot}(\gamma(Q))$

Need to show: $P_{(S_\varepsilon \times T_\bot)_\bot}(\gamma(N),\gamma(Q))$, that is, $\gamma(M) \Uparrow$.

- Since $\gamma(N) \Downarrow V$, we know $P_{S_\varepsilon}(V)$ by forwards preservation.
- In the case $\gamma(Q) \Uparrow$, then we know the pair diverges $(\gamma(N),\ \gamma(Q)) \Uparrow$, and hence $P_{(S_\varepsilon \times T_\bot)_\bot}(\gamma(N),\gamma(Q))$ by our divergence lemma 4.1.

Subcase. $(\alpha = \bot,\ \beta = \bot)$. We have

- $P_{S_\bot}(\gamma(N))$, and
- $P_{T_\bot}(\gamma(Q))$

Need to show: $P_{(S_\bot \times T_\bot)_\bot}(\gamma(N),\gamma(Q))$ i.e, $\gamma(M) \Uparrow$.

- In the case $\gamma(N)$ diverges, then $\gamma(M)$ diverges, so we know $P_{(S_\bot \times T_\bot)_\bot}(\gamma(N),\gamma(Q))$.
- In the case $\gamma(Q)$ diverges, then $\gamma(M)$ diverges, so we know $P_{(S_\bot \times T_\bot)_\bot}(\gamma(N),\gamma(Q))$.

Case(IfzR). In this case M is of shape $\mathsf{ifz}\, N\, \mathsf{then}\, R\, \mathsf{else}\, Q$.

$$(\mathsf{IfzR})\ \frac{\Gamma \vdash R : \mathsf{Nat}_{\alpha}, \qquad \begin{array}{l}\Gamma \vdash R : S_{\beta},\\ \Gamma \vdash Q : S_{\delta},\end{array}}{\Gamma \vdash \mathsf{ifz}\, N\, \mathsf{then}\, R\, \mathsf{else}\, Q : S_{\sigma = \alpha \oplus \beta \oplus \delta},}$$

Need to show $P_{S_\sigma}(\gamma(\mathsf{ifz}\, N\, \mathsf{then}\, R\, \mathsf{else}\, Q)) \equiv P_{S_\sigma}(\mathsf{ifz}\, \gamma(N)\, \mathsf{then}\, \gamma(R)\, \mathsf{else}\, \gamma(Q))$. By the induction hypothesis, we have.

1. $P_{\mathsf{Nat}_\alpha}(\gamma(N))$
2. $P_{S_\beta}(\gamma(R))$
3. $P_{S_\delta}(\gamma(Q))$

Subcase. Case of $\sigma = \alpha \oplus \beta \oplus \delta = \varepsilon$.

In the case the effect is ε for each type, we know N, R and Q all evaluate. Let U, V and W be their values. By forward preservation, we have $P_{\mathsf{Nat}_\varepsilon}(\gamma(U))$, $P_{\mathsf{Nat}_\varepsilon}(\gamma(V))$, $P_{\mathsf{Nat}_\varepsilon}(\gamma(W))$. We know $\gamma(M) \Downarrow$.

Continue by induction on S_σ.

- In the case $S_\sigma = \mathsf{Nat}_\varepsilon$, we have $P_{\mathsf{Nat}_\varepsilon}(\gamma(M))$ which follows from the fact $\gamma(M)$ evaluates.
- In the case $S_\sigma = (A \to B)_\varepsilon$, we must show $P_B(\gamma(M)N')$ for some $P_A(N')$.
  - Suppose $P_A(N')$. By the evaluation rules for (IfzR) expressions, either $\gamma(M)N' \rhd^* VN'$ or $\gamma(M)N' \rhd^* WN'$, depending on whether U is $\underline{0}$ or not.
  - We know $P_B(VN')$ and $P_B(WN')$ by definition of $P_{(A\to B)_\varepsilon}(\gamma(M))$ and the fact $P_{(A\to B)_\varepsilon}(V)$, $P_{(A\to B)_\varepsilon}(W)$.
  - By preservation lemma we get $P_{(A\to B)_\varepsilon}(\gamma(M)N')$ as required.
- In the case $S_\sigma = (A \rightarrowtail B)_\varepsilon$, then we must show $P_A(N')$ supposing $P_B(\gamma(M)N')$. Similar to (AbsR).
  - Suppose $P_B(\gamma(M)N')$. We know that $\gamma(M)N' \Downarrow$ and that it must be that $N' \Downarrow G$ for some value G.
  - We know $P_A(G)$ by definition of $P_{(A\rightarrowtail B)_\varepsilon}(\gamma(M))$. Which gives us $P_A(N')$ by backwards preservation.
- In the case $S_\sigma = (A \times B)_\varepsilon$, we must show $(P_A(\underline{\mathsf{fst}}\, \gamma(M)) \wedge P_B(\underline{\mathsf{snd}}\, \gamma(M))$, given $M \Downarrow (V, W)$.
  - Suppose $P_{(A\times B)_\varepsilon}(\gamma(M))$. By the evaluation rules for (IfzR) expressions, either $\underline{\mathsf{fst}}\, \gamma(M) \rhd^* V$ or $\underline{\mathsf{snd}}\, \gamma(M) \rhd^* W$, in the case U is either $\underline{0}$ or not.
  - We know $P_A(V)$ and $P_B(W)$ by definition of $P_{(A\to B)_\varepsilon}(M)$ and the fact $P_{(A\times B)_\varepsilon}(V)$, $P_{(A\times B)_\varepsilon}(W)$.
  - By the preservation lemma we get $P_{(A\times B)_\varepsilon}(\gamma(M))$ as required.

Subcase. Case of $\alpha \oplus \beta \oplus \delta = \bot$.

In the case the effect is $\bot$ for the overall effect, the whole term diverges or evaluates to a value. This is the case when either N, R or Q diverges. We have that either $P_{\mathsf{Nat}_\bot}(\gamma(N))$, $P_{\mathsf{Nat}_\bot}(\gamma(R))$, or $P_{\mathsf{Nat}_\bot}(\gamma(Q))$. In this case $\gamma(M)$ either diverges or evaluates and we show $P_B(\gamma(M)N')$ for some $P_A(N')$.

Continue by induction on S_σ.

- In the case $S_\sigma = \mathsf{Nat}_\bot$, we have $P_{\mathsf{Nat}_\bot}(M)$ which follows from the when $\gamma(M) \Uparrow$.
- In the case $S_\sigma = (A \to B)_\bot$, we must show $\gamma(M)$ either diverges or evaluates and show $P_B(\gamma(M)N')$ for some $P_A(N')$.
  - We have that $P_{\mathsf{Nat}_\bot}(\gamma(N))$. We show that when $\gamma(N) \Uparrow$ or $\gamma(N) \Downarrow$, then either $\gamma(M) \Uparrow$ or $\gamma(M) \Downarrow$.
  - When $\gamma(N) \Uparrow$ then we know it must be that $\gamma(M) \Uparrow$, and from the divergence lemma 4.1, we know $P_{(A\to B)_\bot}(\gamma(M))$.
  - Otherwise, if $\gamma(N) \Downarrow U$ for some U, we proceed by analysing when $\gamma(R) \Uparrow$ and $\gamma(Q) \Uparrow$.
  - In which case, $\gamma(M) \Uparrow$ whether or not U is $\underline{0}$, and we reach the same conclusion as in the second bullet.

- In the case $\gamma(R) \Downarrow$ and $\gamma(Q) \Downarrow$ then $\gamma(M) \Downarrow$ and we proceed like in the first subcase above.

– In the case $S_\sigma = (A \rightarrowtail B)_\perp$, we must show $\gamma(M)$ either diverges or evaluates and show $P_A(N')$ supposing $P_B(\gamma(M)N')$.
- We have that $P_{\mathsf{Nat}_\perp}(\gamma(N))$. We show that when $\gamma(N) \Uparrow$ or $\gamma(N) \Downarrow$, then either $\gamma(M) \Uparrow$ or $\gamma(M) \Downarrow$.
- When $\gamma(N) \Uparrow$ then we know it must be that $\gamma(M) \Uparrow$, and from the divergence lemma 4.1, we know $P_{(A \rightarrowtail B)_\perp}(\gamma(M))$.
- Otherwise, if $\gamma(N) \Downarrow U$ for some U, we proceed by analysing when $\gamma(R) \Uparrow$ and $\gamma(Q) \Uparrow$.
- In which case, $\gamma(M) \Uparrow$ whether or not U is $\underline{0}$, and we can reach the same conclusion as in the second bullet.
- In the case $\gamma(R) \Downarrow$ and $\gamma(Q) \Downarrow$ then $\gamma(M) \Downarrow$ and we proceed like in the first Subcase.

– In the case $S_\sigma = (A \times B)_\perp$ we must show $\gamma(M)$ can either diverge or evaluate and show $(P_A(\underline{\mathsf{fst}}\,\gamma(M)) \wedge P_B(\underline{\mathsf{snd}}\,\gamma(M))$.
- Suppose $P_{(A\times B)_\perp}(\gamma(M))$.
- In the case $\gamma(N) \Uparrow$ then we know $\gamma(M) \Uparrow$, and from the divergence lemma 4.1, we know $P_{(A\times B)_\perp}(\gamma(M))$.
- Otherwise if $\gamma(N) \Downarrow U$ for some U either $\underline{\mathsf{fst}}\,\gamma(M) \Uparrow$ or $\underline{\mathsf{snd}}\,\gamma(M) \Uparrow$, in the case U is either $\underline{0}$ or not, and hence $\gamma(M) \Uparrow$ and $P_{(A\times B)_\perp}(\gamma(M))$.
- Otherwise if $\gamma(R) \Downarrow$ and $\gamma(Q) \Downarrow$ then $\gamma(M) \Downarrow$ and we proceed like in the first subcase.

□

By taking the term M to be closed in our Effect Progress Theorem 4.5.1, we obtain the effect properties as a corollary, every term in P_{S_α} either evaluates to a value in the case $\alpha = \varepsilon$, or diverges or evaluates to a value in the case $\alpha = \perp$.

4.5.1 Points for consideration

In this subsection, we note some of the challenges faced while undertaking this project.

Fix cases A case that we have omitted due to time includes the Fix case for abstraction. This case was more complex than our non-recursive abstraction cases because it involved working with an induction hypothesis that included the recursive function $f : (S_\alpha \to T_\perp)$ that would satisfy some extended environment γ'. The full induction hypothesis for the (FixsR) case having a form along the lines of $\gamma' \models \Gamma, f : (S_\alpha \to T_\perp)_\varepsilon, x : S_\varepsilon$. Our final goal would have been similar to the 3rd subcase for abstraction where we had to show $P_{(T_\varepsilon \to S_\perp)\varepsilon}(\mathsf{fix}\, f(x).\, \gamma(N))$.

LetR case Another case that was not fully completed due to time constraints was the elimination forms for pairs, (LetR). Introducing the case is fairly straightforward. Suppose we are trying to show $P_{S_\alpha}(\mathsf{let}\,(x, y) = \gamma(N)\,\mathsf{in}\,\gamma(Q))$ for a term M, where $\alpha = \varepsilon$. From our induction hypothesis, we know $P_{(A\times B)_\varepsilon}(\gamma'(N))$ and $P_{S_\varepsilon}(\gamma'(Q))$, where γ' is some extended environment which includes x and y at the correct type. We can instantiate γ' like follows: $\gamma' = \gamma[V/x, W/y]$ where (V, W) is a pair of values. Since we know $P_{(A\times B)_\varepsilon}(\gamma'(M))$, we also know $P_A(V) \wedge P_B(W)$ from the fact M evaluates. We would then continue along the lines of (IfzR), performing an induction over S_α for each type to show $P_{S_\varepsilon}(\gamma'(Q))$.

Predicate Definitions Our unary predicate P, which defined the properties we wished a term to have, often changed throughout the duration of the project. The original definition initially just extended the definition presented by a paper on strong normalisation [8] to account for progress. However, this led to issues with completing the preservation lemma by trying to show certain properties our assumptions did not give us enough information on. One such case was the forward part for preservation, where we tried to show $P_{\mathsf{Nat}_\perp}(N)$ where $M \rhd N$. Our induction hypothesis only gave us that $P_{\mathsf{Nat}_\perp}(M)$, which meant that M either steps or is a value, and we already know M steps to N, leaving us not being able to proceed in the proof. For the backwards cases, the original definition was less of an issue since showing $P_{\mathsf{Nat}_\perp}(M)$ was trivial given $M \rhd N$, which satisfied M making a step. This led to our current definition, where we specify $\perp$ involves the possibility of M literally diverging rather than just making a step in the traditional sense.

Well-typedness and Divergence Another issue that was tricky to resolve included proving if a term was in the divergence relation for elimination forms, for example, showing (1) $\forall N.\, P_{T_\perp}((\lambda x.\, M)N) \implies P_{S_\perp}(N)$. Although this case initially seemed to be covered by our divergence lemma 4.1, there was the problem of possible type mismatches when trying to prove an implication that involved some form of the predicate implication in (1). One case where this problem might arise was in trying to prove the above: if the output of the application diverges or evaluates, the given argument must have diverged or evaluated, i.e. $(S_\perp \rightharpoonup T_\perp)_\varepsilon$. The issue stems from not knowing if N is indexed at the correct type, $S_\perp$. When trying to show this property is preserved under elimination, we experimented by generalising the divergence lemma to remove the requirement of the term being well-typed. However, this would also have meant removing the well-typed requirement from each of our predicate definitions P as well, involving a significant change to what is proven by the relation definition.

An alternate way to describe logical predicates involves using value and expression interpretations. These are described in [8] as functions from types to the power set of closed values. It has been noted that, at least in value and expression interpretations, well-typedness is not required in the predicate to prove type safety. Otherwise, if it were included, preservation would need to be proved for certain proofs to pass.

Despite the outlook for this issue in (1), the problem may not have been too involved from the start. Our requirement of terms to be well-typed in the relation informs us of the shape of M, which we could use with the predicate assumption to inform us that an application is well-typed, from which we can use inversion to infer the type of $P_{S_\perp}(N)$. So, for now, we can decide to keep the well-typed requirement for the proof unless further evidence convinces us otherwise.

Chapter 5

Conclusion

5.1 Summary

In Chapter 3, we began by re-introducing the two-sided system and introduced the basic definitions for effects annotations in the language.

We then introduced the typing rules modified with effects. We elaborate on the changed inference rules, explaining why each rule has the shape it does.

We provided examples of deriving proof trees for terms with effect annotated types. This allowed us to show how types with certain effect properties are true.

Chapter 4 covers the soundness quality that we desire for our inference rules and constitutes the main contributions of the project. This was shown through progress and preservation. Progress ensured that well-typed terms that satisfied our logical predicate would not get stuck. Preservation guaranteed that well-typed terms would be preserved under the steps of evaluation.

We also explain the alterations we made to the definition of progress and preservation to ensure the theorems could be proven in both the forward and backward directions.

In this chapter, we also motivated the use of a logical predicate from logical relations to prove our soundness property. The main reason for this was that we could not prove an application evaluated without a stronger induction hypothesis, which the logical predicate allowed us to define.

We defined the properties we desired of terms over the structure of the types in our system and ensured the conditions were preserved under elimination forms.

Finally, we proved our Effects Progress Theorem to obtain our soundness property that every term either diverged or evaluated a value, and every step taken preserves the well-typed property.

5.2 Future Work

In this section, we discuss possible extensions and future avenues for research relating to this project.

5.2.1 Crashing

Crashing $\lightning$ is another effect that we could prove for the current system. We say a term *crashes*, *gets stuck*, or *goes wrong* just if it reduces to a stuck term. Building upon the rules of our current system, adding crashing would involve proving that the effect is preserved under the forward and backward reduction lemma and showing the additional sub-cases for each inference rule.

5.2.2 Left rules

From the last chapter, we have the soundness result for the right rules of the two-sided system. The next immediate stage would be to try and tackle proving the soundness of effects for the left rules. This would include reproducing definitions for the meaning of a predicate (for example, Q_{S_α}) and defining what properties a term M satisfying this predicate has and if the condition is preserved by elimination forms (if applicable).

For basic type of Nat, we could define a predicate for Q_{Nat_α} like follows:

$$Q_{\mathsf{Nat}_\alpha}(M) \quad ::= \quad M : \mathsf{Nat}_\alpha \vdash \wedge\,(\alpha = \varepsilon \wedge\ (M \Downarrow V,\, V \notin \text{numeral } \underline{n}) \vee M \Uparrow \vee M \text{ crashes})$$

Where M crashes means that the term gets stuck. In the above predicate, refuting term M has the type Nat_ε means that M either evaluates to non-numeric value (e.g. an abstraction) or diverges or crashes. Note that we mean the complemented effect of α when attached to the predicate Q_{S_α} above.

In the current semantics of the system, any typed term on the left simply refutes the fact that the term evaluates to that type. With the addition of our annotated effects, these left rules may also allow for refuting terms that do not diverge or crash.

We would extend this over the rest of the system types and prove the preservation property for forward and backward reduction. As an example with the forward case, suppose we are trying to prove the following as a continuation from the previous case with Nat:

$$\text{(Forward)} \quad M : S_\alpha \vdash \wedge\, M \rhd N \wedge Q_{S\alpha}(M) \implies Q_{S\alpha}(N)$$

This might be proven as follows:

Case($\mathbf{Nat}_\alpha$).

1. $\alpha = \varepsilon$: Suppose $Q_{\mathsf{Nat}_\varepsilon}(M)$.

   Need to show $Q_{\mathsf{Nat}_\varepsilon}(N)$ where $M \rhd N$.

   - We have $M \Downarrow V$, $V \notin \underline{n}$ or $M \Uparrow$ or M crashes.
   - Since CBV evaluation is deterministic, we know M and N must evaluate to the same value.
   - Hence it must also be that $N \Downarrow V \notin \underline{n}$.
   - Otherwise, when $M \Uparrow$ we know $N \Uparrow$ since $M \rhd N$.
   - It is also clear that if M crashes, then it reduces to the stuck term N.
   - And from this we can conclude $Q_{\mathsf{Nat}_\varepsilon}(N)$.

We know M crashes just if M reduces to a stuck term (N in this case). In the backward direction for crashing, if N crashes, then we know M will also reduce to a stuck term from $M \rhd N$.

After completing preservation for the rest of the types, we would then go on to prove effect progress for the left rules by induction over each typing derivation on the left, giving us the following theorem.

Theorem 5.1 (Effect Progress on the Left)**.** $\Gamma,\, M : S_\alpha \vdash \wedge\, \gamma \models \Gamma \implies Q_{S_\alpha}(\gamma(M))$

5.2.3 Equivalence with (CompR)

From the previous two-sided system described in the original paper, we know that the rule (CompR) is unsound in normal CBV semantics, as explained in the introduction Section 1.1. The next step would be to use our newfound precision in expressing effects to allow us to redefine the (CompR) rule in a form which is semantically sound.

$$\frac{\Gamma,\, M : S_\varepsilon \vdash}{\Gamma \vdash M : S^c{}_{\perp\!\!\!/}} \,(\mathsf{CompR})$$

5.2.4 Subtyping and subeffecting

Another extension that could be considered involves subtyping or subeffecting in the in our effect system. From our effect definitions, we see that ε could be considered a *subeffect* of $\perp$ since terms with type $S_\perp$ either diverge or evaluate to a value whereas terms annotated with S_ε evaluate only. So, having no effect could internalise a 'stronger' meaning than divergence. One way to formalise this might be using the principle of safe substitution [10] so that we could use ε safely in any context in which we used $\perp$, denoted by $\varepsilon \subseteq \perp$. However, this proves to be unsound, as it would allow us to convert $\underline{\mathsf{div}}$ with type $\mathsf{Nat}_\perp$ to that of Nat_ε, which would now assert that $\underline{\mathsf{div}}$ has evaluated. It appears the subsumption rule introduced in our background Section 2.1 may not be sound with our current effect definitions.

We may want to instead extend our system with some sort of subtyping for our arrow functions in light of Example 3.4. The (AppR) rule allows us to deduce that the abstraction $\lambda x.\, 2 : (\mathsf{Nat}_\perp \to \mathsf{Nat}_\varepsilon)_\varepsilon$ must have had the type effect $\mathsf{Nat}_\perp$ for the input. However, in CBV, we would desire for the premises of the application to instead have a type of $(\mathsf{Nat}_\varepsilon \to \mathsf{Nat}_\varepsilon)_\varepsilon$, since the diverging input would be evaluated before the function is applied. This may call for some sort of subtyping, allowing us to replace occurrences

of $S_\varepsilon \to T_\alpha$ with $S_\perp \to T_\perp$. This could let us convert the types after deducing the premises, but may also involve changing the annotations for (AppR).

As we only proved the effect system for two additional effects at the start, subtyping seemed excessive in that its addition could involve considering multiple new subtyping rules on top of the current ones and require further changes to our soundness proof. It was not immediately obvious how to resolve this mismatch in the premises, and it would require more research for this particular case.

Bibliography

[1] Steven Ramsay and Charlie Walpole. Ill-Typed Programs Don't Evaluate. *Proceedings of the ACM on Programming Languages*, 8(POPL), January 2024.

[2] David K. Gifford, Pierre Jouvelot, Mark A. Sheldon, and James W. O'Toole. Report on the FX-91 Programming Language:. Technical report, Defense Technical Information Center, Fort Belvoir, VA, February 1992.

[3] David K. Gifford and John M. Lucassen. Integrating functional and imperative programming. In *Proceedings of the 1986 ACM conference on LISP and functional programming*, LFP '86, pages 28–38, New York, NY, USA, August 1986. Association for Computing Machinery.

[4] Flemming Nielson and Hanne Riis Nielson. Type and Effect Systems. volume 1710, pages 114–136, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg. Book Title: Correct System Design Series Title: Lecture Notes in Computer Science.

[5] Robert Harper. *Practical Foundations for Programming Languages*. Cambridge University Press, USA, 2nd edition, March 2016.

[6] Adrian Sampson. Proving Type Safety for STLC, 2016.

[7] Frank Pfenning. Lecture Notes on Progress and Preservation. 2021.

[8] Lau Skorstengaard. An Introduction to Logical Relations. 2015.

[9] Andrew Myers. Strong Normalization and Logical Relations, April 2013.

[10] Benjamin C. Pierce. *Types and Programming Languages*. The MIT Press, 1st edition, January 2002.

Appendix A

Lemmas

Lemma A.1 (Canonical Forms)**.** *Suppose* $\vdash V : S_\alpha$ *and* V *is a value, then*

- *If* S *is a* Nat*, then* V *is a numeral* $\underline{n}$*.*
- *If* S *is a* $A \to B$*, then* V *is an abstraction, i.e.* $\lambda x.M : B$ *for some* x *and* M*.*
- *If* S *is a* $A \times B$*, then* V *is a pair* (V_1, V_2) *for some* V_1 *value and* V_2 *value.*
- *If* S *is a* $A \rightarrowtail B$*, then* V *is an abstraction i.e.* $\lambda x.M : A$ *for some* x *and* M*.*

Proof. By inspection of the inference rules in 3.1.

- If S is a Nat, then the only rule which lets us give a value this type is (ZeroR).
- If S is a $A \to B$, then the only rule which lets us give a value this type is (FixsR) or (AbsR).
- If S is a $A \times B$, then the only rule which lets us give a value this type is (PairR).
- If S is a $A \rightarrowtail B$, then the only rule which lets us give a value this type is (FixnR) or (AbnR).

□