Applying Types as Abstract Interpretation to a Language with Dynamic Dispatch

MSc Thesis *(Afstudeerscriptie)*

written by

Johannes Emerich
(born April 7th, 1985 in Augsburg, Germany)

under the supervision of **Dr. Tijs van der Storm** and **Prof. Dr. Benno van den Berg**, and submitted to the Board of Examiners in partial fulfillment of the requirements for the degree of

MSc in Logic

at the *Universiteit van Amsterdam.*

Date of the public defense:
January 30th, 2015

Members of the Thesis Committee:
Prof. Dr. Ronald de Wolf (chair)
Dr. Tijs van der Storm (supervisor)
Prof. Dr. Benno van den Berg (supervisor)
Prof. Dr. Jan van Eijck
Prof. Dr. Paul Klint
Dr. Wouter Swierstra

Institute for Logic, Language and Computation

Abstract

We are concerned with an application of the "types as Abstract Interpretation" perspective of [Cou97] to the problem of typing overloaded operator symbols in a simple applicative language with dynamic dispatch. We begin by a development of abstract semantics, or type systems, for λ_{C}, a language without overloading. The distinction between *parametric* and *ad-hoc* type polymorphism is then introduced, where a particular challenge arising from the *ad-hoc* variant is stressed. λ_{K}, an extension of λ_{C} with support for overloading definitions at the user level serves to illustrate the problem of *ad-hoc* polymorphism, and is presented together with a type system designed to support some of its forms. To prove soundness of said type system, the abstract interpretation development of type systems for λ_{C} is extended to λ_{K} and the new type system is shown to be a sound abstraction in the sense of [Cou97].

To the wonderful people of the ILLC.

Acknowledgments

I thank first and foremost my supervisors: Tijs van der Storm for suggesting the topic of type systems and permitting me to improvise quite freely on the theme, and Benno van den Berg for his continued support and patience. I thank the members of my thesis committee, Ronald de Wolf, who agreed to chair my defense, and Jan van Eijck, Paul Klint and Wouter Swierstra, who agreed to read my thesis.

I thank my family for their supply of love and firewood, and all of my friends for extending their support and being understanding of my monkish existence, but especially Ulrich Swoboda for being a true and hilarious mensch on *WhatsApp*. I am grateful that Ignas Vyšniauskas became both inspiring collaborator and friend. My heart belongs to the beautiful people I was lucky to meet in the *Master of Logic*.

Contents

1 Introduction **4**
- 1.1 Outline of the Thesis 5
- 1.2 Notational Debt 6

2 An Untyped Lambda Calculus with Runtime Errors **7**
- 2.1 Syntax for λ_{C} 7
- 2.2 A Denotational Semantics 8

3 Abstract Interpretation and Type Inference **10**
- 3.1 A Cautious Reaction to Undecidable Program Properties 10
- 3.2 Fundamentals of Abstract Interpretation 11
  - 3.2.1 Program Properties 11
  - 3.2.2 Capturing Abstraction Formally 12
- 3.3 Type Inference as Abstract Interpretation 13
  - 3.3.1 Abstract Semantics and Sound Type Systems 13
  - 3.3.2 An Idealized Type System: Type Collecting Semantics . . 14
  - 3.3.3 Towards Implementable Systems: Church/Curry Polytypes 17

4 An Example: Bounded Polymorphism **20**
- 4.1 The Problem 20
  - 4.1.1 Parametric Polymorphism 21
  - 4.1.2 Ad-hoc Polymorphism 22
- 4.2 A Language With Dynamic Dispatch 23
  - 4.2.1 Type Language for Type System $\mathbb{T}^{\mathrm{T}}$ 23
  - 4.2.2 The Language λ_{K} 24
  - 4.2.3 Translation Without Type Inference 25
- 4.3 A Type System for λ_{K} 27
  - 4.3.1 Unification with Constrained Variables 27
  - 4.3.2 A Rule-based Definition of $\mathbb{T}^{\mathrm{T}}$ 30

5 Abstract Interpretation of Overloading **31**
- 5.1 Type Collecting Semantics for Programs with Overloading 31
- 5.2 Church/Curry Polytype Semantics for Programs with Overloading 35
  - 5.2.1 Abstraction from Type Collecting Semantics 36

5.3 The Type System $\mathbb{T}^{\mathrm{T}}$ as an Abstract Semantics 38
5.3.1 Abstraction from Church/Curry Polytype Semantics . . . 40

6 Related Work **43**
6.1 Bounded Polymorphism 43
6.2 Dynamic Dispatch 45
6.3 Abstract Interpretation and Type Analysis 45

7 Conclusion **47**

Appendix A Some Errors in [Cou97] **49**
A.1 The Soundness of $\mathbf{T}^{\mathrm{Co}}[\![\cdot]\!]$ 49

Notation **55**

Bibliography **57**

Chapter 1

Introduction

> The real power of type theory is that well-formedness of the formalised expressions implies logical and mathematical correctness of the original content.
>
> —Nederpelt & Geuvers, *Type Theory and Formal Proof*

> I wonder who it was defined man as a rational animal. It was the most premature definition ever given.
>
> —Lord Henry in Oscar Wilde's *The Picture of Dorian Gray*

The two above citations may serve to introduce the background motivation of this thesis: To work towards a reconciliation of the mathematical perspective on programming languages and what we might—perhaps perilously—call *human nature*. While the citation on the "power of type theory" is concerned mainly with the role of type theory in mathematics, similar sentiments abound in the field of programming language theory. Among the benefits commonly attributed to static type checking are [Car96]:

- Improved runtime performance through elimination of safety checks,
- improved development time through bug prevention,
- improved maintainability of large codebases,
- facilitation of advanced language features, and
- improved compilation performance.

However, these empirical claims are in need of substantiation through evidence, which especially in the case of the effect on development time, bug prevention and maintainability is hard to gather—and indeed, this is seldom done [Han10]. What is easy to observe on the other hand is the popularity of programming languages with no or very weak static type systems.

In this context, there are two readings of Lord Henry's above proclamation. For one, the opinion that human folly is to blame for the low rate of adoption of obviously superior methods enjoys a certain popularity among programming language theorists. This is not our intended reading. On the contrary, we suggest to consider the option that there may be a certain wisdom in the popular preference for languages which do not impose a strong static type discipline.

To make this suggestion seem less ludicrous we refer to the long-running discussion of the role of formalism in mathematics, where repeated arguments have been made to stress the importance of intuition and gradual discovery in the development of new mathematical content[1]. In a similar fashion as mathematics, the authoring of computer programs is a gradual process of exploration in a problem space that is often not completely understood at the onset.

Recent years have seen increased interest in programming languages which aim to find a good trade-off between static and dynamic type systems in *gradual typing* [RCH12]. Languages such as *ActionScript*, *TypeScript* and *Dart* still integrate type checking into the compilation process, but allow partial typings, that is, they allow the translation of even maltyped expressions into executable programs. *Racket*, a dialect of *Lisp*, goes even further in externalizing its optional type system into a library which is independent of the compiler [THSAC$^+$11] to create a "programmable programming language" [PLT14] in which language users may assume control over type-based program analysis and optimization.

This motivates the idea of relating type analysis to the more general discipline of program analysis. Our point of departure is Patrick Cousot's 1997 paper *Types as Abstract Interpretation* [Cou97], in which various type *inference* systems for a simple dynamic language are compared from within the framework of *abstract interpretation*. The result is a "hierarchy of type systems, which is part of the lattice of abstract interpretations of the untyped lambda-calculus" [Cou97, p. 316], that is, the study of type systems can potentially be connected to the study of a wider range of program analyses.

1.1 Outline of the Thesis

Our goal is to extend Cousot's treatment by considering a simple form of ad-hoc polymorphism. For this purpose, we cautiously extend the language of [Cou97] by constructs for defining operations that behave differently on different *types* of inputs. We introduce a new type system for this language which is inspired by earlier work by Stefan Kaes [Kae88, Kae05] and prove its soundness by abstraction from earlier abstract interpretation developments.

Chapter 2 introduces syntax and denotational semantics of a simple applicative language λ_{C} which is subject to abstract interpretation in the rest of the thesis.

Chapter 3 provides some background on and basic notions of abstract interpretation before stating definitions of type systems as abstract semantics. The

[1] For direct claims see for example [Kli74] or [Tal13], for a historical account [Gra08].

rest of the chapter is dedicated to presenting two of the type systems introduced in [Cou97] with slight adaptation to match the language from Chapter 2.

Chapter 4 introduces the notion of type polymorphism which appears as *parametric* and *ad-hoc* polymorphism. To illustrate challenges of the latter form, we define an extension to syntax and semantics of λ_{C}, introducing a simple form of overloading. We define a new type system for this language, which possesses a type inference algorithm as an extension of Damas-Milner type inference with an enhanced notion of unification with sorted type variables. We prove that the unification procedure produces most general unifiers.

Chapter 5 extends the type systems from Chapter 3 to accommodate the extended language and uses these results to prove soundness of the new type system by showing it to be an abstraction of the prior systems.

Chapter 6 discusses related work on bounded polymorphism, dynamic languages with dynamic dispatch, and type inference using abstract interpretation.

Chapter 7 concludes with a discussion of our contributions.

1.2 Notational Debt

As a warning to readers we list some of the abuses of notation we are aware of being guilty of in this thesis. This section can be skimmed or skipped entirely.

- We will often partially apply functions which may or may not be curried.
- We will rely on a simple variant of lambda calculus notation in the meta language to denote objects in the denotational semantics:
  - $\Lambda u. \cdots$ denotes a function
  - $\Lambda u_1 \ldots u_n. \cdots$ is short for $\Lambda u_1. \ldots \Lambda u_n. \cdots$
  - $\mathsf{if}\ \cdots\ \mathsf{then}\ \cdots\ \mathsf{else}\ \cdots$ is used to specify conditionals

Chapter 2

An Untyped Lambda Calculus with Runtime Errors

> It may be that in all her phrases stirred
> The grinding water and the gasping wind;
> But it was she and not the sea we heard.
>
> —Wallace Stevens, *The Idea of Order At Key West*

This brief chapter serves to present λ_{C}, a fairly standard variation on the untyped lambda calculus together with a denotational semantics. The language is designed to resemble the one employed in [Cou97], with only superficial changes intended to improve readability. Familiarity with basic domain theory is assumed and the reader is invited to skim or skip the natural language paragraphs with the exception of Definition 2.2.1 which is important for later comprehension.

2.1 Syntax for λ_{C}

Our language adds a separate syntactic form for fixpoint definitions in order to increase applicability of a variety of type systems, which may be unable to accommodate the Y-combinator directly. The language furthermore possesses primitives for integers and integer addition, as well as a conditional expression with a zero test. We present an abstract syntax with an unspecified representation of the integers but may use obvious literals such as `101` or `-5` if needed.

$$\begin{aligned} \mathtt{x}, \mathtt{f}, \ldots &\in \mathbb{X} : \text{program variables} \\ z &\in \mathbb{Z} : \text{integers} \\ e &\in \mathbb{E} : \text{program expressions} \end{aligned}$$

$$
\begin{aligned}
e ::= {} & \mathtt{x} \mid \lambda \mathtt{x}.e \mid e_1 e_2 \mid (e) \mid \\
& \mu \mathtt{f}.\lambda \mathtt{x}.e \mid \mathtt{let\ x} = e_1 \ \mathtt{in}\ e_2 \mid \\
& z \mid e_1 + e_2 \mid \\
& \mathtt{ifz}\ e_1\ \mathtt{then}\ e_2\ \mathtt{else}\ e_3
\end{aligned}
$$

2.2 A Denotational Semantics

Because we will later introduce an extended language which requires a more intricate notion of program environments, we parametrize the definition of program semantics by program environments.

Definition 2.2.1 (Program Semantics)**.** An $\mathscr{E}$-program semantics is a *semantic domain* $\mathbb{S}_{\mathscr{E}} = \mathscr{E} \to \mathbb{U}$ together with a *semantic function* $\mathbf{S}_{\mathscr{E}} : \mathbb{E} \to \mathbb{S}_{\mathscr{E}}$.

In this and the following chapter, we will be concerned with an $\mathbb{R}$-program semantics, where $\mathbb{R}$ is a simple mapping from identifiers to values. In Section 4.2 we will introduce an extension which supports operator overloading via dynamic dispatch and requires more intricate program environments for this purpose. We will refer to $\mathbb{S}_{\mathbb{R}}$ as simply $\mathbb{S}$ and $\mathbf{S}_{\mathbb{R}}$ as simply $\mathbf{S}$.

We present the semantic domain for program denotations, which is unchanged from [Cou97, p. 316] and relies on domain theoretic definitions [GS90, DP02]. The bottom element $\bot$ represents non-terminating computations, $\mathcal{D}_\bot$ is the lift of $\mathcal{D}$ by $\bot$, $\uparrow(\cdot) : \mathcal{D} \to \mathcal{D}_\bot$ injects into the lift, and $\downarrow(\cdot) : \mathcal{D}_\bot \nrightarrow \mathcal{D}$ extracts from the lift where possible. $[\mathcal{D}_1 \to \mathcal{D}_2]$ denotes the domain of functions from $\mathcal{D}_1 \to \mathcal{D}_2$ which are strict on $\bot$ and ✘.

$$
\begin{aligned}
\mathbb{Z} &= \{\ldots, -2, -1, 0, 1, 2, \ldots\} && \text{integers} \\
\mathbb{W} &:= \{\mathbf{wrong}\} && \text{wrong} \\
\text{✘} &:= \uparrow(\mathbf{wrong}) :: \mathbb{W}_\bot && \text{injected wrong} \\
\mathbb{U} &\cong \mathbb{W}_\bot \oplus \mathbb{Z}_\bot \oplus [\mathbb{U} \to \mathbb{U}]_\bot && \text{values} \\
\mathbb{R} &:= \mathbb{X} \to \mathbb{U} && \text{program environments} \\
\mathbb{S} &:= \mathbb{R} \to \mathbb{U} && \text{program denotations}
\end{aligned}
$$

$\mathbb{U}$ is ordered by an *information* or *computational* ordering $\sqsubseteq$ such that $\bot$ is $\sqsubseteq$-least. We write $\cdot :: \mathcal{D}$ for tagged injections of values from $\mathcal{D}$ into $\mathbb{U}$.

We now explain the meaning of λ_{C} by giving its denotational semantics. The everywhere undefined function $\varepsilon_\bot$ is defined as $\uparrow(\Lambda u.\bot) :: [\mathbb{U} \to \mathbb{U}]_\bot$. We will make repeated use of the functions $\mathbf{lfp}^{\leq}_x$ and $\mathbf{gfp}^{\leq}_x$ which compute the $\leq$-least fixpoint greater than or equal to x, and the $\leq$-greatest fixpoint less than or equal to x, respectively.

$$
\begin{aligned}
\mathbf{S}[\![\mathtt{x}]\!] &:= \Lambda R.R(\mathtt{x}) \\
\mathbf{S}[\![\lambda \mathtt{x}.e]\!] &:= \Lambda R.\uparrow(\Lambda u.\mathsf{if}\ u \in \{\bot, \text{✘}\}\ \mathsf{then}\ u \\
&\qquad\qquad \mathsf{else}\ \mathbf{S}[\![e]\!] R[\mathtt{x} \leftarrow u]) :: [\mathbb{U} \to \mathbb{U}]_\bot
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{S}[\![e_1 e_2]\!] &:= \Lambda R.\mathsf{if}\ \bot \in \{\mathbf{S}[\![e_1]\!]R, \mathbf{S}[\![e_2]\!]R\}\ \mathsf{then}\ \bot \\
&\qquad \mathsf{else\ if}\ \mathbf{S}[\![e_1]\!]R = f :: [\mathbb{U} \to \mathbb{U}]_\bot\ \mathsf{then} \\
&\qquad\quad \downarrow(f)(\mathbf{S}[\![e_2]\!]R) \\
&\qquad \mathsf{else}\ \text{✘} \\
\mathbf{S}[\![\mu \mathtt{x}.\lambda \mathtt{f}.e]\!] &:= \Lambda R.\mathbf{lfp}^{\sqsubseteq}_{\varepsilon_\bot}(\Lambda u.\mathbf{S}[\![\lambda \mathtt{x}.e]\!]R[\mathtt{f} \leftarrow u]) \\
\mathbf{S}[\![\mathtt{let\ x\ =}\ e_1\ \mathtt{in}\ e_2]\!] &:= \Lambda R.\mathbf{S}[\![(\lambda x.e_2)e_1]\!]R \\
\mathbf{S}[\![z]\!] &:= \Lambda R.\uparrow(z) :: \mathbb{Z}_\bot \\
\mathbf{S}[\![e_1 + e_2]\!] &:= \Lambda R.\mathsf{if}\ \bot \in \{\mathbf{S}[\![e_1]\!]R, \mathbf{S}[\![e_2]\!]R\}\ \mathsf{then}\ \bot \\
&\qquad \mathsf{else\ if}\ \mathbf{S}[\![e_i]\!]R = \uparrow(z_i) :: \mathbb{Z}_\bot\ \mathsf{then} \\
&\qquad\quad \uparrow(z_1 + z_2) :: \mathbb{Z}_\bot \\
&\qquad \mathsf{else}\ \text{✘} \\
\mathbf{S}[\![\mathtt{ifz}\ e_1\ \mathtt{then}\ e_2\ \mathtt{else}\ e_3]\!] &:= \Lambda R.\mathsf{if}\ \mathbf{S}[\![e_1]\!]R = \bot\ \mathsf{then}\ \bot \\
&\qquad \mathsf{else\ if}\ \mathbf{S}[\![e_1]\!]R = \uparrow(z) :: \mathbb{Z}_\bot\ \mathsf{then} \\
&\qquad\quad \mathbf{S}[\![\mathsf{if}\ z = 0\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3]\!]R \\
&\qquad \mathsf{else}\ \text{✘}
\end{aligned}
$$

Chapter 3

Abstract Interpretation and Type Inference

3.1 A Cautious Reaction to Undecidable Program Properties

> I says to myself, I reckon a body that ups and tells the truth when he is in a tight place is taking considerable many resks, though I ain't had no experience, and can't say for certain; but it looks so to me, anyway; and yet here's a case where I'm blest if it don't look to me like the truth is better and actuly SAFER than a lie.
>
> —Mark Twain, *The Adventures of Huckleberry Finn*

A basic result of computability theory is due to Henry G. Rice [Ric53] (statement adapted from [Odi99, p. 150]):

Theorem 3.1.1 (Rice's Theorem)**.** *A class of partial recursive functions A is computable if and only if it is trivial, i.e. either empty or containing all partial recursive functions.*

As a consequence, any non-trivial program property is undecidable. In particular, there is no general procedure that is both sound and complete with respect to a given property: Any total general procedure must by necessity err on at least one side, and will falsely attribute the property to programs which do not possess it, will falsely deny the property of programs that do possess it, or both.

From this dreadful situation, a slight but important information advantage can be gained by establishing of a certain procedure that it errs on only one side. In the formal methods community it has long been a popular choice to sacrifice completeness in exchange for soundness, so that procedures will never falsely attribute a property, but instead will deny it in error at least some of the

time. In this way, incorrect programs can always be rejected. The objective, then, is to devise procedures with a vanishing number of *false alarms*.

Abstract interpretation [CC77, AH87] (often stylized as *Abstract Interpretation*) is a theoretical framework for the formal study and systematic development of abstractions of program semantics, with the aim of facilitating safe reasoning by means of sound abstraction, where the semantics of a sound abstraction "includes"—in a sense yet to be made precise—the *real* semantics. In this manner it can be ensured that a sound abstract semantics safely approximates properties of the original semantics.

3.2 Fundamentals of Abstract Interpretation

In addition to the engineerial evolution of the practical methods derived from abstract interpretation, several variations on the fundamental idea have been proposed in scholarly works. A handbook article by Abramsky and Hankin [AH87], a text book by Nielson et al. [NNH99], and surveys by the Cousots [Cou00, CC14] may serve to provide a partial overview.

For focus and brevity, we will introduce abstract interpretation in the context of our program semantics for λ_{C} and λ_{K}, where the theoretical background is drawn from mostly [Cou97], and other resources are only referred to as needed. We again (2.2.1) parametrize our definitions by program environments $\mathscr{E}$ to be able to treat λ_{C} and λ_{K} uniformly.

3.2.1 Program Properties

The following definitions generalize those of [Cou97, p. 317] to variants parametrized by program environments.

Definition 3.2.1 (Semantic Properties)**.** An *$\mathscr{E}$-semantic property* $P \in \mathbb{P}_{\mathscr{E}} := \mathcal{P}(\mathscr{E} \to \mathbb{U})$ is a set of program denotations which can be understood as the collection of programs for which the property at hand holds.

The set $\mathbb{P}_{\mathscr{E}} := \mathcal{P}(\mathbb{S}_{\mathscr{E}})$ forms a complete lattice $\langle \mathbb{P}_{\mathscr{E}}, \subseteq, \emptyset, \mathbb{S}_{\mathscr{E}}, \cup, \cap, .^{\complement} \rangle$, where $.^{\complement}$ is complementation in $(\mathscr{E} \to \mathbb{U})$, and where the set operations may be interpreted as logical operations on properties.

We immediately introduce our first abstract semantics (3.3.1), which merely converts program denotations to program properties.

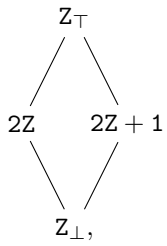
Definition 3.2.2 (Standard Collecting Semantics)**.** The *$\mathscr{E}$-standard collecting semantics* $\mathbf{C}[\![\cdot]\!] : \mathbb{E} \to \mathbb{P}_{\mathscr{E}}$ is defined as

$$\mathbf{C}[\![e]\!] := \{\mathbf{S}_{\mathscr{E}}[\![e]\!]\}.$$

By lifting program denotations to properties through the collecting semantics, we are able to make precise the notion of abstraction.

3.2.2 Capturing Abstraction Formally

For a very simple example of abstraction, consider the complete lattice formed by the $\mathcal{P}(\mathbb{Z})$ preordered by set inclusion, $\langle \mathcal{P}(\mathbb{Z}), \subseteq, \emptyset, \mathbb{Z}, \cup, \cap \rangle$. This structure is abstracted by the much simpler $\langle \mathtt{Z}, \leq_{\mathtt{Z}}, \mathtt{Z}_\bot, \mathtt{Z}_\top, \vee_{\mathtt{Z}}, \wedge_{\mathtt{Z}} \rangle$ given by



where we interpret the objects in the lattice as the empty set, the evens, the odds, and the integers.

For each $Z \subseteq \mathbb{Z}$ we have an abstract—in this case finite, and possibly less precise—representation in $\mathtt{Z}$, $\emptyset$ as $\mathtt{Z}_\bot$, $\{1, 2\}$ as $\mathtt{Z}_\top$, $\{\ldots, -4, -2, 0, 2, 4, \ldots\}$ as $2\mathtt{Z}$, etc. For some Z we can move to an abstract representation in $\mathtt{Z}$ and back to Z, in other cases such as the set $\{1, 2\}$, we incur a loss of precision by converting the best abstract representation back into a set of numbers. However, it is always true that any Z will be included in the concrete set representation of Z's best abstraction.

The situation is that one structure acts as a concrete representation of certain objects, which the other structure represents more concisely but with less precision. To capture the conversions between abstract and concrete representations, we introduce a pair of maps $\langle \alpha, \gamma \rangle$, of which α takes concrete objects to their abstract representation, and γ goes the opposite way from abstract to concrete. In our example, $\alpha : \mathcal{P}(\mathbb{Z}) \to \mathtt{Z}$,

$$\alpha(Z) := \begin{cases} \mathtt{Z}_\bot & \text{if } Z = \emptyset \\ 2\mathtt{Z} & \text{if } Z \subseteq 2\mathbb{Z} \\ 2\mathtt{Z}+1 & \text{if } Z \subseteq 2\mathbb{Z}+1 \\ \mathtt{Z}_\top & \text{o.w.} \end{cases}$$

and $\gamma : \mathtt{Z} \to \mathcal{P}(\mathbb{Z})$,

$$\gamma(z) := \begin{cases} \emptyset & \text{if } z = \mathtt{Z}_\bot \\ 2\mathbb{Z} & \text{if } z = 2\mathtt{Z} \\ 2\mathbb{Z}+1 & \text{if } z = 2\mathtt{Z}+1 \\ \mathbb{Z} & \text{if } z = \mathtt{Z}_\top. \end{cases}$$

Note that, for any $Z \in \mathbb{Z}, z \in \mathtt{Z}$,

$$\alpha(Z) \leq_{\mathtt{Z}} z \Leftrightarrow Z \subseteq \gamma(z),$$

and hence, because α is monotone,

$$Z \subseteq \gamma(\alpha(Z)),$$

which captures the idea of safe approximation.

This *adjoint situation* is an instance of an isotone Galois connection.

Definition 3.2.3 (Galois Connection)**.** For posets X, Y, we call $\langle \alpha, \gamma \rangle$ a *Galois connection* between X and Y, written $\langle X, \leq \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle Y, \leq \rangle$, if for all $x \in X, y \in Y$

$$\alpha(x) \leq y \Leftrightarrow x \leq \gamma(y).$$

In the general case we call α the *left adjoint* of γ, and conversely γ α's *right adjoint*. In practice we will refer to α and γ as the *abstraction* and *concretization* function, respectively. We call the Galois connection *isotone* if the maps are order-preserving, and *antitone* if they are order-reversing.

Definition 3.2.4 (Galois Insertion)**.** For posets X, Y, we call a Galois connection $\langle \alpha, \gamma \rangle$ a *Galois insertion* if α is a surjection.

We will introduce further terminology and standard results as required. A good general introduction can be found in [DP02].

3.3 Type Inference as Abstract Interpretation

3.3.1 Abstract Semantics and Sound Type Systems

Definition 3.3.1 (Abstract Semantics)**.** An *abstract semantics* $\langle \mathscr{P}^\natural, \leq^\natural, \mathscr{S}^\natural[\![\cdot]\!] \rangle$ consists of a poset $\langle \mathscr{P}^\natural, \leq^\natural \rangle$ together with a function $\mathscr{S}^\natural[\![\cdot]\!] : \mathbb{E} \to \mathscr{P}^\natural$. We may interpret the triple as a set of abstract properties with logical implication and an abstract semantic function mapping program expressions to program properties. Where there is no threat of ambiguity, we may refer to an abstract semantics by its set of abstract properties.

We call an abstract semantics an *abstract interpreter* if its poset of abstract properties is computer representable and its abstract semantic function is computable.

Definition 3.3.2 (Sound Abstraction)**.** For two abstract semantics, $\langle \mathscr{P}^\natural, \leq^\natural, \mathscr{S}^\natural[\![\cdot]\!] \rangle$, $\langle \mathscr{P}^\sharp, \leq^\sharp, \mathscr{S}^\sharp[\![\cdot]\!] \rangle$, we call $\mathscr{P}^\natural$ an *abstraction* of the *concrete semantics* $\mathscr{P}^\sharp$ if there is a *concretization function*, that is, a monotone function $\gamma : \mathscr{P}^\natural \to \mathscr{P}^\sharp$ such that for all $e \in \mathbb{E}$ we have $\mathscr{S}^\sharp[\![e]\!] \leq^\sharp \gamma(\mathscr{S}^\natural[\![e]\!])$.

We call an abstract semantics *$\mathscr{E}$-sound* if it is an abstraction of the $\mathscr{E}$-standard collecting semantics $\langle \mathbb{P}_{\mathscr{E}}, \subseteq, \mathbf{C}[\![\cdot]\!] \rangle$.

Proposition 3.3.3 ([Cou97])**.** *An abstraction of a sound abstract semantics is sound.*

Definition 3.3.4 (Sound Type System)**.** A *$\mathscr{E}$-sound type system* $\langle \mathbb{T}, \leq, \mathbf{T}[\![\cdot]\!], \gamma, \mathcal{E} \rangle$ consists of a sound abstract semantics $\langle \mathbb{T}, \leq, \mathbf{T}[\![\cdot]\!] \rangle$ via concretization function $\gamma : \mathbb{T} \to \mathbb{P}_{\mathscr{E}}$, together with an *admissible environments function* $\mathcal{E} : \mathbb{T} \to \mathcal{P}(\mathscr{E})$ mapping types to program environments, such that

$$\forall T \in \mathbb{T} : \forall \phi \in \gamma(T) : \forall \hat{R} \in \mathcal{E}(T) : \phi(\hat{R}) \neq \text{✘}.$$

Definition 3.3.5 (Typability)**.** We call a program $e \in \mathbb{E}$ *typable* in $\hat{R} \in \mathscr{E}$ if there is a $\mathscr{E}$-sound type system $\mathcal{T} = \langle \mathbb{T}, \leq, \mathbf{T}[\![\cdot]\!], \gamma, \mathcal{E} \rangle$ for which $\hat{R} \in \mathcal{E}(\mathbf{T}[\![e]\!])$.

From the preceding definitions we can immediately conclude the following

Proposition 3.3.6 ("Typable Programs Can't Go Wrong")**.** *For any $e \in \mathbb{E}$, if e is typable in $\hat{R} \in \mathscr{E}$, then $\mathbf{S}[\![e]\!]\hat{R} \neq$ ✘.*

3.3.2 An Idealized Type System: Type Collecting Semantics

The type collecting semantics given in [Cou97, p. 323] is the initial abstraction of program properties which approximates program properties by very precise types. All later type systems inherit their soundness from this semantics by abstraction. In introducing this semantics, Cousot suggests it to be a candidate for an answer to the question "What is a type system", by virtue of the fact that it "is more precise than the reduced product of all existing type systems" [Cou97, p. 323]—a claim of an empirical nature which we feel unable to decide the veridity of. Because our development relies on an adaptation of Cousot's construction to prove soundness of a type system for our extension of Cousot's language, this section as well as the next largely restates results from [Cou97].

We first define the semantic domain of *collecting polytypes*, $\mathbb{P}^{\mathrm{Co}}$, then a pair of maps $\langle \alpha^{\mathrm{Co}}, \gamma^{\mathrm{Co}} \rangle$ that clarifies their relation to program semantics.

Definition 3.3.7.

$$\begin{aligned}
\mathbb{P}^{Co^0} &:= \{\bot^{Co}, \texttt{int}\} && \text{basic types} \\
\mathbb{P}^{Co^{\delta+1}} &:= \mathbb{P}^{Co^\delta} \cup \mathcal{P}(\mathbb{P}^{Co} \times \mathbb{P}^{Co}) && \text{form function types} \\
\mathbb{P}^{Co^\lambda} &:= \bigcup_{\delta<\lambda} \mathbb{P}^{Co^\delta} && \text{flatten limit ordinal} \\
t \in T \subseteq \mathbb{P}^{\mathrm{Co}} &:= \bigcup_{\delta \text{ ordinal}} \mathbb{P}^{Co^\delta} && \text{flatten hierarchy}
\end{aligned}$$

We explain the meaning of collecting polytypes by giving a concretization function to collections in the denotational semantics $\gamma_1^{Co} : \mathbb{P}^{\mathrm{Co}} \to \mathcal{P}(\mathbb{U})$.

$$\gamma_1^{Co^0}(\bot^{Co}) := \{\bot\}, \gamma_1^{Co^0}(\texttt{int}) := \mathbb{Z}_\bot$$
$$\gamma_1^{Co^\delta}(t_1 \to t_2) := \{\uparrow(\phi) :: [\mathbb{U} \to \mathbb{U}]_\bot \mid$$

$$\phi \in [\mathbb{U} \to \mathbb{U}] \wedge \forall u \in \gamma_1^{Co^\delta}(t_1) : \phi(u) \in \gamma_1^{Co^\delta}(t_2)\} \cup \{\bot\}$$

$$\gamma_1^{Co^{\delta+1}}(t) := \gamma_1^{Co^\delta}(t)$$

$$\gamma_1^{Co^{\delta+1}}(T) := \bigcap_{t_1 \to t_2 \in T} \gamma_1^{Co^\delta}(t_1 \to t_2)$$

$$\gamma_1^{\mathrm{Co}}(\bot^{\mathrm{Co}}) := \mathbb{U} \qquad \text{where } \bot^{\mathrm{Co}} := \emptyset$$

$$\gamma_1^{Co^\lambda}(t) := \gamma_1^{Co^\delta}(t) \qquad \text{for } \delta < \lambda$$

$$\gamma_1^{\mathrm{Co}}(t) := \gamma_1^{Co^\delta}(t) \qquad \text{for } t \in \mathbb{P}^{Co^\delta}$$

An order and corresponding equivalence ($\equiv^{\mathrm{Co}}$) on collecting polytypes is defined using the concretization to denotations,

$$t_1 \leq^{\mathrm{Co}} t_2 :\Leftrightarrow \gamma_1^{\mathrm{Co}}(t_1) \subseteq \gamma_1^{\mathrm{Co}}(t_2),$$

such that $\langle \mathbb{P}^{\mathrm{Co}}/\equiv^{\mathrm{Co}}, \leq^{\mathrm{Co}} \rangle$ is a partial order. In the following, we write $\mathbb{P}^{\mathrm{Co}}_{\equiv}$ for $\mathbb{P}^{\mathrm{Co}}/\equiv^{\mathrm{Co}}$.

A meet operation on collecting polytypes can be defined by

$$\bigwedge^{Co} T := \begin{cases} \bot^{\mathrm{Co}} & \text{if } \bot^{\mathrm{Co}} \in T \text{ or } T \text{ contains several type constructors} \\ \texttt{int} & \text{if } T = \{\texttt{int}\} \\ \emptyset^{\mathrm{Co}} & \text{if } T = \emptyset \\ \bigcup T & \text{if } T \subseteq \mathcal{P}(\mathbb{P}^{\mathrm{Co}}_{\equiv} \times \mathbb{P}^{\mathrm{Co}}_{\equiv}) \end{cases}$$

γ_1^{Co} preserves meets [Cou97, p. 324], so a definition of meets of collecting polytypes allows direct derivation of an abstraction function,

$$\alpha_1^{\mathrm{Co}}(U) := \bigwedge^{Co}\{t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \,|\, U \subseteq \gamma_1^{\mathrm{Co}}(t)\}.$$

This allows us to define a join,

$$\bigvee^{Co} T := \alpha_1^{\mathrm{Co}}(\textstyle\bigcup_{t \in T} \gamma_1^{\mathrm{Co}}[t]),$$

so that $\langle \mathbb{P}^{\mathrm{Co}}_{\equiv}, \leq^{\mathrm{Co}}, \bot^{\mathrm{Co}}, \emptyset^{\mathrm{Co}}, \bigvee^{Co}, \bigwedge^{Co} \rangle$ is a complete lattice.

Next we extend this abstraction to collections of environments which are approximated by type environments $H \in \mathbb{H}^{\mathrm{Co}} := \mathbb{X} \to \mathbb{P}^{\mathrm{Co}}_{\equiv}$, with

$$\alpha_2^{\mathrm{Co}}(\mathcal{R}) := \Lambda \mathtt{x}.\alpha_1^{\mathrm{Co}}(\{R(\mathtt{x}) \,|\, R \in \mathcal{R}\})$$
$$\gamma_2^{\mathrm{Co}}(H) := \{R \in \mathbb{R} \,|\, \forall \mathtt{x} \in \mathbb{X} : R(\mathtt{x}) \in \gamma_1^{\mathrm{Co}}(H(\mathtt{x}))\}$$
$$H_1 \dot{\leq}^{\mathrm{Co}} H_2 :\Leftrightarrow \forall \mathtt{x} \in \mathbb{X} : H_1(\mathtt{x}) \leq^{\mathrm{Co}} H_2(\mathtt{x}),$$

which gives us a Galois insertion

$$\langle \mathcal{P}(\mathbb{R}), \subseteq, \emptyset, \mathbb{R}, \cap, \cup \rangle \underset{\alpha_2^{\mathrm{Co}}}{\overset{\gamma_2^{\mathrm{Co}}}{\leftrightarrows}} \langle \mathbb{H}^{\mathrm{Co}}, \dot{\leq}^{\mathrm{Co}}, \dot{\bot}^{\mathrm{Co}}, \dot{\emptyset}^{\mathrm{Co}}, \dot{\bigvee}^{Co}, \dot{\bigwedge}^{Co} \rangle.$$

To complete the sequence of abstractions, we abstract program properties $P \in \mathbb{P}_{\mathbb{R}}$ by typings $\theta \in \mathbb{T}^{\mathrm{Co}} := \mathbb{H}^{\mathrm{Co}} \to \mathbb{P}^{\mathrm{Co}}_{\equiv}$, where

$$\begin{aligned}
\alpha^{\mathrm{Co}}(P) &:= \Lambda H.\alpha_1^{\mathrm{Co}}(\{\phi(R) \,|\, R \in \gamma_2^{\mathrm{Co}}(H) \wedge \phi \in P\}) \\
\gamma^{\mathrm{Co}}(\theta) &:= \{\phi \,|\, \forall H \in \mathbb{H}^{\mathrm{Co}} : \forall R \in \gamma_2^{\mathrm{Co}}(H) : \phi(R) \in \gamma_1^{\mathrm{Co}}(\theta(H))\} \\
\theta_1 \ddot{\leq}^{\mathrm{Co}} \theta_2 &:\Leftrightarrow \forall H \in \mathbb{H}^{\mathrm{Co}} : \theta_1(H) \leq^{\mathrm{Co}} \theta_2(H),
\end{aligned}$$

such that

$$\langle \mathbb{P}_{\mathbb{R}}, \subseteq, \emptyset, \mathbb{S}, \cap, \cup \rangle \underset{\alpha^{\mathrm{Co}}}{\overset{\gamma^{\mathrm{Co}}}{\leftrightarrows}} \langle \mathbb{T}^{\mathrm{Co}}, \ddot{\leq}^{\mathrm{Co}}, \ddot{\bot}^{\mathrm{Co}}, \ddot{\emptyset}^{\mathrm{Co}}, \ddot{\bigvee}^{Co}, \ddot{\bigwedge}^{Co} \rangle$$

is a Galois connection.

We are now ready to state the semantic function $\mathbf{T}^{\mathrm{Co}}[\![\cdot]\!] : \mathbb{E} \to \mathbb{P}^{\mathrm{Co}}_{\equiv}$ and prove that $\mathbb{T}^{\mathrm{Co}}$ is a sound abstraction. Our definition of $\mathbf{T}^{\mathrm{Co}}[\![\cdot]\!]$ deviates from [Cou97] in two cases which seem to contain errors in the original presentation, such that the intended properties do not hold as stated. For a discussion of the original and altered definitions, we refer to Appendix A.1.

$$\begin{aligned}
&\mathbf{T}^{\mathrm{Co}}[\![\mathtt{x}]\!] := \Lambda H.H(\mathtt{x}) \\
&\mathbf{T}^{\mathrm{Co}}[\![\lambda \mathtt{x}.e]\!] := \\
&\quad \Lambda H. \bigwedge\nolimits^{\mathrm{Co}} \left\{ \{t \to \mathbf{T}^{\mathrm{Co}}[\![e]\!]H[\mathtt{x} : t]\} \,\middle|\, t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \setminus \{\emptyset^{\mathrm{Co}}\} \right\} \\
&\mathbf{T}^{\mathrm{Co}}[\![e_1 e_2]\!] := \\
&\quad \Lambda H. \bigwedge\nolimits^{\mathrm{Co}} \left\{ t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \,\middle|\, \mathbf{T}^{\mathrm{Co}}[\![e_1]\!]H \leq^{\mathrm{Co}} \{\mathbf{T}^{\mathrm{Co}}[\![e_2]\!]H \to t\} \right\} \\
&\mathbf{T}^{\mathrm{Co}}[\![\mu \mathtt{f}.\lambda \mathtt{x}.e]\!] := \\
&\quad \Lambda H.\mathbf{lfp}_{\{\emptyset^{\mathrm{Co}} \to \bot^{\mathrm{Co}}\}} \Lambda t.\mathbf{T}^{\mathrm{Co}}[\![\lambda \mathtt{x}.e]\!]H[\mathtt{f} : t] \\
&\mathbf{T}^{\mathrm{Co}}[\![\mathtt{let\ x} = e_1 \ \mathtt{in}\ e_2]\!] := \\
&\quad \Lambda H. \bigwedge\nolimits^{\mathrm{Co}} \left\{ \mathbf{T}^{\mathrm{Co}}[\![e_2]\!]H[\mathtt{x} : t] \,\middle|\, t = \mathbf{T}^{\mathrm{Co}}[\![e_1]\!]H \neq \emptyset^{\mathrm{Co}} \right\} \\
&\mathbf{T}^{\mathrm{Co}}[\![z]\!] := \mathtt{int} \\
&\mathbf{T}^{\mathrm{Co}}[\![e_1 + e_2]\!] := \mathtt{int} \vee^{\mathrm{Co}} (\mathbf{T}^{\mathrm{Co}}[\![e_1]\!]H \vee^{\mathrm{Co}} \mathbf{T}^{\mathrm{Co}}[\![e_2]\!]H) \\
&\mathbf{T}^{\mathrm{Co}}[\![\mathtt{ifz}\ e_1\ \mathtt{then}\ e_2\ \mathtt{else}\ e_3]\!] := \\
&\quad \Lambda H.\ \mathsf{if}\ \mathbf{T}^{\mathrm{Co}}[\![e_1]\!]H = \bot^{\mathrm{Co}}\ \mathsf{then} \\
&\qquad \bot^{\mathrm{Co}} \\
&\qquad \mathsf{else\ if}\ \mathbf{T}^{\mathrm{Co}}[\![e_1]\!]H = \mathtt{int}\ \mathsf{then} \\
&\qquad\quad \mathbf{T}^{\mathrm{Co}}[\![e_2]\!]H \vee^{\mathrm{Co}} \mathbf{T}^{\mathrm{Co}}[\![e_3]\!]H \\
&\qquad \mathsf{else}\ \emptyset^{\mathrm{Co}}
\end{aligned}$$

We state Cousot's results without proof, as we will reconstruct parts of the proofs for our language extension below. For additional details, the interested reader is again referred to the appendix, A.1.

Proposition 3.3.8. *Any $\ddot{\leq}^{\mathrm{Co}}$-upper approximation of the best abstraction of the standard collecting semantics is sound, that is, for any $e \in \mathbb{E}, H \in \mathbb{H}^{\mathrm{Co}}, t \in \mathbb{P}^{\mathrm{Co}}_{\equiv}$:*

$$\alpha^{\mathrm{Co}}(\mathbf{C}[\![e]\!])H \leq^{\mathrm{Co}} t \Leftrightarrow \forall R \in \gamma_2^{\mathrm{Co}}(H) : \mathbf{S}[\![e]\!]R \in \gamma_1^{\mathrm{Co}}(t).$$

Proposition 3.3.9. *For all $e \in \mathbb{E}$, $\mathbf{T}^{\mathrm{Co}}[\![e]\!]$ is monotone in that for any $H_1, H_2 \in \mathbb{H}^{\mathrm{Co}}$ we have*

$$H_1 \dot{\leq}^{\mathrm{Co}} H_2 \Rightarrow \mathbf{T}^{\mathrm{Co}}[\![e]\!]H_1 \leq^{\mathrm{Co}} \mathbf{T}^{\mathrm{Co}}[\![e]\!]H_2.$$

Proposition 3.3.10. *The type collecting semantics $\mathbf{T}^{\mathrm{Co}}[\![\cdot]\!]$ is an $\ddot{\leq}^{\mathrm{Co}}$-upper approximation of $\alpha^{\mathrm{Co}}(\cdot)$, as for any $e \in \mathbb{E}$*

$$\alpha^{\mathrm{Co}}(\mathbf{C}[\![e]\!]) \ddot{\leq}^{\mathrm{Co}} \mathbf{T}^{\mathrm{Co}}[\![e]\!].$$

It remains to devise an admissible type environment function $\mathcal{E}^{\mathrm{Co}}$, which may be defined as

$$\mathcal{E}^{\mathrm{Co}}(\theta) := \bigcup\{\gamma_2^{\mathrm{Co}}(H) \mid H \in \mathbb{H}^{\mathrm{Co}} \wedge \theta(H) \neq \emptyset^{\mathrm{Co}}\},$$

and we get

Corollary 3.3.11. *$\langle \mathbb{T}^{\mathrm{Co}}, \ddot{\leq}^{\mathrm{Co}}, \mathbf{T}^{\mathrm{Co}}[\![\cdot]\!], \gamma^{\mathrm{Co}}, \mathcal{E}^{\mathrm{Co}} \rangle$ is a sound type system.*

3.3.3 Towards Implementable Systems: Church/Curry Polytypes

With the next abstraction, we move closer towards realizable type systems by virtue of less precise, and finite, types. However, *polymorphic* types, which appear in type environments, are still potentially infinite structures and may not be implemented directly. Under some additional restrictions on polytypes, it is possible to represent them finitely by means of a further Herbrand abstraction [Cou97, p. 321f], where infinite sets of types are represented as finite type *schemes*. This leads to a family of variations around the type system first introduced by Robin Milner [Mil78], which are discussed in the closing section of [Cou97]. We will develop a similar abstraction in the context of our language extension.

Cousot introduces the "À la Church/Curry Polytype Semantics" as a refinement of a type system [Cou97, p. 317] derived as an idealized development of Church's simply-typed lambda calculus [Chu40] and Curry's type inference for combinatory logic [CF58]. As in the systems of Church and Curry, types assigned to program expressions are monomorphic, but in the refinement identifiers in type environments may be associated with polymorphic types, which are sets of monomorphic type terms.

We again restate definitions and results from [Cou97], synthesizing the developments of [Cou97, Section 6 & 16]. The domain of types $\mathbb{T}^{\mathrm{PC}}$ is defined as

follows:

$$
\begin{aligned}
m \in \mathbb{M}^{\mathrm{PC}}, m &::= \mathtt{int} \mid m_1 \to m_2 \\
p \in \mathbb{P}^{\mathrm{PC}} &:= \mathcal{P}(\mathbb{M}^{\mathrm{PC}}) \\
H \in \mathbb{H}^{\mathrm{PC}} &:= \mathbb{X} \to \mathbb{P}^{\mathrm{PC}} \\
\theta \in \mathbb{I}^{\mathrm{PC}} &:= \mathbb{H}^{\mathrm{PC}} \times \mathbb{M}^{\mathrm{PC}} \\
T \in \mathbb{T}^{\mathrm{PC}} &:= \mathcal{P}(\mathbb{I}^{\mathrm{PC}})
\end{aligned}
$$

Cousot defines a direct Galois connection between $\mathbb{T}^{\mathrm{PC}}$ and $\mathbb{P}_{\mathbb{R}}$, which we omit as the actual proof of soundness is via $\mathbb{P}^{\mathrm{Co}}$ in both Cousot's and our development. We therefore proceed directly to defining a semantic function $\mathbf{T}^{\mathrm{PC}}[\![\cdot]\!] : \mathbb{E} \to \mathbb{T}^{\mathrm{PC}}$, wherein $\mathbb{M}^{\mathrm{PC}} \to \mathbb{M}^{\mathrm{PC}} := \{m_1 \to m_2 \,|\, m_1, m_2 \in \mathbb{M}^{\mathrm{PC}}\}$.

$$
\begin{aligned}
&\mathbf{T}^{\mathrm{PC}}[\![\mathtt{x}]\!] := \left\{\langle H, m\rangle \;\middle|\; m \in H(\mathtt{x}) \wedge H \in \mathbb{H}^{\mathrm{PC}}\right\} \\
&\mathbf{T}^{\mathrm{PC}}[\![\lambda \mathtt{x}.e]\!] := \left\{\langle H, m_1 \to m_2\rangle \;\middle|\; \langle H[\mathtt{x} : \{m_1\}], m_2\rangle \in \mathbf{T}^{\mathrm{PC}}[\![e]\!]\right\} \\
&\mathbf{T}^{\mathrm{PC}}[\![e_1 e_2]\!] := \\
&\quad \left\{\langle H, m_2\rangle \;\middle|\; \langle H, m_1 \to m_2\rangle \in \mathbf{T}^{\mathrm{PC}}[\![e_1]\!] \wedge \langle H, m_1\rangle \in \mathbf{T}^{\mathrm{PC}}[\![e_2]\!]\right\} \\
&\mathbf{T}^{\mathrm{PC}}[\![\mathtt{let\ x} = e_1 \ \mathtt{in}\ e_2]\!] := \\
&\quad \left\{\langle H, m_2\rangle \;\middle|\; \exists p_1 \neq \emptyset : \big(\forall m_1 \in p_1 : \langle H, m_1\rangle \in \mathbf{T}^{\mathrm{PC}}[\![e_1]\!]\right. \\
&\qquad \left.\wedge\ \langle H[\mathtt{x} : p_1], m_2\rangle \in \mathbf{T}^{\mathrm{PC}}[\![e_2]\!]\big)\right\} \\
&\mathbf{T}^{\mathrm{PC}}[\![\mu \mathtt{f}.\lambda \mathtt{x}.e]\!] := \\
&\quad \left\{\langle H, m\rangle \;\middle|\; m \in \mathbf{gfp}_{\mathbb{M}^{\mathrm{PC}} \to \mathbb{M}^{\mathrm{PC}}} \Lambda p.\{m' | \langle H[\mathtt{f} : p], m'\rangle \in \mathbf{T}^{\mathrm{PC}}[\![\lambda \mathtt{x}.e]\!]\}\right\} \\
&\mathbf{T}^{\mathrm{PC}}[\![z]\!] := \left\{\langle H, \mathtt{int}\rangle \;\middle|\; H \in \mathbb{H}^{\mathrm{PC}}\right\} \\
&\mathbf{T}^{\mathrm{PC}}[\![e_1 + e_2]\!] := \left\{\langle H, \mathtt{int}\rangle \;\middle|\; \langle H, \mathtt{int}\rangle \in \mathbf{T}^{\mathrm{PC}}[\![e_1]\!] \cap \mathbf{T}^{\mathrm{PC}}[\![e_2]\!]\right\} \\
&\mathbf{T}^{\mathrm{PC}}[\![\mathtt{ifz}\ e_1\ \mathtt{then}\ e_2\ \mathtt{else}\ e_3]\!] := \\
&\quad \left\{\langle H, m\rangle \;\middle|\; \langle H, \mathtt{int}\rangle \in \mathbf{T}^{\mathrm{PC}}[\![e_1]\!] \wedge \langle H, m\rangle \in \mathbf{T}^{\mathrm{PC}}[\![e_2]\!] \cap \mathbf{T}^{\mathrm{PC}}[\![e_3]\!]\right\}
\end{aligned}
$$

With this we are ready for the definition of a concretization function $\gamma^{\mathrm{Po}} : \mathbb{T}^{\mathrm{PC}} \to \mathbb{T}^{\mathrm{Co}}$ which will allow us to obtain a soundness result for Church/Curry polytypes.

$$
\begin{aligned}
\gamma_1^{\mathrm{Po}} &: \mathbb{M}^{\mathrm{PC}} \to \mathbb{P}^{\mathrm{Co}}_{\equiv} \\
\gamma_1^{\mathrm{Po}}(\mathtt{int}) &:= \mathtt{int} \\
\gamma_1^{\mathrm{Po}}(m_1 \to m_2) &:= \{\gamma_1^{\mathrm{Po}}(m_1) \to \gamma_1^{\mathrm{Po}}(m_2)\} \\
\gamma_2^{\mathrm{Po}} &: \mathbb{P}^{\mathrm{PC}} \to \mathbb{P}^{\mathrm{Co}}_{\equiv} \\
\gamma_2^{\mathrm{Po}}(p) &:= \bigwedge\nolimits^{\mathrm{Co}} \{\gamma_1^{\mathrm{Po}}(m) \,|\, m \in p\} \\
\gamma_3^{\mathrm{Po}} &: \mathbb{H}^{\mathrm{PC}} \to \mathbb{H}^{\mathrm{Co}} \\
\gamma_3^{\mathrm{Po}}(H) &:= \gamma_2^{\mathrm{Po}} \circ H
\end{aligned}
$$

$$\gamma_4^{\mathrm{Po}} : \mathbb{I}^{\mathrm{PC}} \to \mathbb{T}^{\mathrm{Co}}$$
$$\gamma_4^{\mathrm{Po}}(\langle H, m\rangle) := \Lambda H'.\bigwedge\nolimits^{\mathrm{Co}}\{\gamma_1^{\mathrm{Po}}(m) \mid H' \dot{\leq}^{\mathrm{Co}} \gamma_3^{\mathrm{Po}}(H)\}$$
$$\gamma^{\mathrm{Po}} : \mathbb{T}^{\mathrm{PC}} \to \mathbb{T}^{\mathrm{Co}}$$
$$\gamma^{\mathrm{Po}}(T) := \ddot{\bigwedge}^{\mathrm{Co}}\{\gamma_4^{\mathrm{Po}}(\langle H, m\rangle) \mid \langle H, m\rangle \in T\},$$

in which $\ddot{\bigwedge}^{\mathrm{Co}} X := \Lambda H.\bigwedge^{\mathrm{Co}}\{\theta(H) \mid \theta \in X\}$.

Proposition 3.3.12 ([Cou97, p. 326])**.** *Together with the abstraction* $\alpha^{\mathrm{Po}}(\theta) = \{\langle H, m\rangle \mid \theta(\gamma_3^{\mathrm{Po}}(H)) \leq^{\mathrm{Co}} \gamma_2^{\mathrm{Po}}(\{m\})\}$*,* γ^{Po} *forms a Galois connection*

$$\langle \mathbb{T}^{\mathrm{Co}}, \ddot{\leq}^{\mathrm{Co}}, \ddot{\perp}^{\mathrm{Co}}, \ddot{\emptyset}^{\mathrm{Co}}, \ddot{\bigvee}^{\mathrm{Co}}, \ddot{\bigwedge}^{\mathrm{Co}}\rangle \underset{\alpha^{\mathrm{Po}}}{\overset{\gamma^{\mathrm{Po}}}{\leftrightarrows}} \langle \mathbb{T}^{\mathrm{PC}}, \supseteq, \mathbb{H}^{\mathrm{PC}} \times \mathbb{M}^{\mathrm{PC}}, \emptyset, \cap, \cup\rangle.$$

We will again record the soundness result for $\mathbb{T}^{\mathrm{PC}}$ without proof as we will consider some details below, but for later reference, we first state the following

Proposition 3.3.13. *For any* $m_1, m_2 \in \mathbb{M}^{\mathrm{PC}}, \gamma_1^{\mathrm{Po}}(m_1) \leq^{\mathrm{Co}} \gamma_1^{\mathrm{Po}}(m_2)$ *implies* $m_1 = m_2$*.*

Lemma 3.3.14. *The Church/Curry polytype semantics is a sound abstraction, as for all* $e \in \mathbb{E} : \mathbf{T}^{\mathrm{Co}}[\![e]\!] \ddot{\leq}^{\mathrm{Co}} \gamma^{\mathrm{Po}}(\mathbf{T}^{\mathrm{PC}}[\![e]\!])$*.*

As mentioned in introducing the Church/Curry polytype semantics, polymorphic types present the challenge of presenting infinite types finitely. We will now step out of the abstract interpretation mindset to consider the particular problem of *ad-hoc* as opposed to *parametric* polymorphism, which poses additional difficulties by generating a greater variety of type term shapes, complicating representation.

Chapter 4

An Example: Bounded Polymorphism

Nous devons obtenir le laissez-passer A38.

—*Les Douze Travaux d'Astérix*

4.1 The Problem

The ability of operators in programming languages to act on a variety of structurally distinct operands is commonly referred to as *polymorphism*. A dictionary definition of *polymorphism* is "the quality or state of existing in or assuming different forms" [Mer15]. Within computer science, a popular reference defines *polymorphic languages* by their characteristic feature that "some values and variables may have more than one *type*" [CW85, p. 4] (emphasis added). That is, the multiplicity of forms associated with individual values is revealed only when looking through typed glasses[1].

Polymorphism in programming languages is itself at least bimorphic, as it appears in *parametric* and *ad-hoc* form. This distinction goes back to Christopher Strachey's 1967 lecture notes (republished as [Str00]), but is summarized more succinctly in [CW85, p. 4]. Therein, an operator's polymorphism is understood as *parametric* if it operates uniformly across operands of a family of types, where the members of this family exhibit structural commonalities. It is understood as *ad-hoc* if the operator operates on operands of a family of types whose family resemblance can not be captured in the type structure (and may be altogether unsystematic).

We see, then, that parametrically polymorphic operators are monomorphic as values, but with polymorphic types, while ad-hoc polymorphic operators are

[1] We exclude from consideration the related but distinct notion of *polymorphism* in object-oriented languages.

polymorphic in both their value and associated types. The structural commonalities of types arising from parametric polymorphism facilitate their systematic and automatic treatment in programming language technology, whereas the lack of regularity in the types arising from ad-hoc polymorphism present a variety of challenges. We will briefly the principle idea of an early and classic treatment of parametric polymorphism ([Mil78]), before discussing particularities of the challenges presented by ad-hoc polymorphism.

4.1.1 Parametric Polymorphism

In his classic paper *A Theory of Type Polymorphism in Programming*, Robin Milner intended to present a solution to the problem of harmoniously combining "[a] widely employed style of programming" with a language-inherent type discipline that would allow language compilers "to find rather inscrutable bugs" [Mil78, p. 348] automatically. By devising a type system and language that allowed for the automatic inference of types from bare expressions, both programming style and type safety could be preserved. Although our later development builds on the work of Milner, especially by making use of the type inference procedure described in [DM82], we will not go into details of this method. Instead, we will discuss intent and effect of this approach to type polymorphism.

According to Milner in 1978, "the polymorphism present in a program is a natural outgrowth of the primitive polymorphic operators which appear to exist in every programming language", listing assignment, functional application and list processing operations as examples of such primitives. This allows language users to define "procedures which work well on objects of a wide variety (e.g. on lists of atoms, integers, or lists)" [Mil78, p. 348f]. Note that this particular case is a picture-book example of the structural commonality that was attributed to parametric polymorphism in our discussion above. Consider the recursive definition of a simple function designed to compute the length of a list:

```
len = μf.λl. if is-empty l then 0 else (f (tail l)) + 1
```

Here, `is-empty` and `tail` are assumed to be primitive operations on lists, which are polymorphic in the sense that while they require operands to be structures of a certain shape—namely lists—it is of no import to the purpose of these operators what the list is a list of. We may say that for any type `a`, `is-empty` is of type $\texttt{list(a)} \to \texttt{bool}$ and `tail` is of type $\texttt{list(a)} \to \texttt{list(a)}$. That is, the family of types associated with `is-empty` consists of a set

$$\left\{ \begin{array}{l} \texttt{list(int)} \to \texttt{bool}, \texttt{list(bool)} \to \texttt{bool}, \ldots, \\ \texttt{list(list(int))} \to \texttt{bool}, \texttt{list(list(bool))} \to \texttt{bool}, \ldots, \\ \vdots \end{array} \right\},$$

the infinite family $\{\texttt{list(a)} \to \texttt{bool}\}_{\texttt{a} \in T}$ indexed by the set T of all types.

It is quite obvious from the family expression that the object-level use of variables in type expressions would allow for an accurate finite representation of the types associated with `is-empty`. This is precisely the idea behind [Mil78], where type variables in type *schemes* are treated as implicitly universally quantified, such that we may read the type term $\texttt{list(a)} \to \texttt{bool}$ as $\forall \texttt{a}.\texttt{list(a)} \to \texttt{bool}$. Use of Robinson's unification algorithm [Rob65] for first-order terms ensures an effective mechanism for checking whether a particular type is an instance of a particular type scheme, and thereby a member of the represented family of types.

Milner's type system is designed to preserve generality as much as possible when combining the types of subexpressions to obtain the type of an expression, such that `len` above may inherit the generality of its constituents `is-empty` and `tail`, to possess polymorphic type $\texttt{list(a)} \to \texttt{int}$.

4.1.2 Ad-hoc Polymorphism

We come to the ad-hoc form of polymorphism, where we begin by considering an example. Like Milner, we first assume polymorphism to be a consequence of polymorphic primitives, but we will consider a variant of the addition operation which exhibits *ad-hoc* polymorphism, which we explain by a denotational semantics[2]:

$$\begin{aligned}\mathbf{S}'[\![e_1{+}e_2]\!] := \Lambda R.\mathsf{if}\ \bot \in \{\mathbf{S}'[\![e_1]\!]R, \mathbf{S}'[\![e_2]\!]R\}\ \mathsf{then}\ \bot\\ \mathsf{else\ if}\ \mathbf{S}'[\![e_i]\!]R = \uparrow(z_i) :: \mathbb{Z}_\bot\ \mathsf{then}\\ \uparrow(z_1 + z_2) :: \mathbb{Z}_\bot\\ \mathsf{else\ if}\ \mathbf{S}'[\![e_i]\!]R = \uparrow(b_i) :: \mathbb{B}_\bot\ \mathsf{then}\\ \uparrow(b_1 \vee b_2) :: \mathbb{B}_\bot\\ \mathsf{else}\ \text{✘}\end{aligned}$$

The + operation now possesses two functions as alternative *forms*: One computing the sum of integers, the other computing the join of boolean values[3]. The actual function implementation to *dispatch* to is chosen dynamically based on runtime analysis of the actual paramters. Correspondingly, it has two types: $\texttt{int} \to \texttt{int} \to \texttt{int}$ and $\texttt{bool} \to \texttt{bool} \to \texttt{bool}$. We note already that this set can not be compressed in the same manner as above: $\texttt{a} \to \texttt{a} \to \texttt{a}$ would allow too many instances. However, as the collection of types associated with + is finite, this does not present a problem for representation in itself.

This form of ad-hoc polymorphism is referred to as *overloading*, of which the addition operation is maybe the canonical example (usually with integers and floating point numbers). But there is no guarantee for addition to be overloaded in such benign manner only. Consider a variation which is defined as before,

[2] We assume some additional value domains as required.

[3] In fact, the definition suggests *four* forms: a constant function returning bottom, the aforementioned functions on integers and booleans, and a constant function returning the runtime error.

but with an additional implementation for list concatenation:

$$\begin{aligned}\mathbf{S}'[\![e_1{+}e_2]\!] := \Lambda R.\mathsf{if}\ \bot \in \{\mathbf{S}'[\![e_1]\!]R, \mathbf{S}'[\![e_2]\!]R\}\ \mathsf{then}\ \bot \\ \mathsf{else\ if}\ \mathbf{S}'[\![e_i]\!]R = \uparrow(z_i) :: \mathbb{Z}_\bot\ \mathsf{then} \\ \uparrow(z_1 + z_2) :: \mathbb{Z}_\bot \\ \mathsf{else\ if}\ \mathbf{S}'[\![e_i]\!]R = \uparrow(b_i) :: \mathbb{B}_\bot\ \mathsf{then} \\ \uparrow(b_1 \vee b_2) :: \mathbb{B}_\bot \\ \mathsf{else\ if}\ \mathbf{S}'[\![e_i]\!]R = \uparrow(l_i) :: \mathbb{L}_\bot\ \mathsf{then} \\ \uparrow(\mathbf{concat}(l_1, l_2)) :: \mathbb{L}_\bot \\ \mathsf{else}\ \text{✘}\end{aligned}$$

The types of + can now be given as $\{\texttt{int} \to \texttt{int} \to \texttt{int}, \texttt{bool} \to \texttt{bool} \to \texttt{bool}\} \cup \{\texttt{list(a)} \to \texttt{list(a)} \to \texttt{list(a)}\}_{\texttt{a}\in T}$, for which again a finite representation is needed. Milner considered overloading to be "somewhat orthogonal" to parametric polymorphism [Mil78, p. 349], which as we will see later is not quite accurate. For the moment however, we will look for some middle-ground to allow partial reconciliation of overloading with Milner's method for type inference.

4.2 A Language With Dynamic Dispatch

There is a variety of possible ways in which we could extend the language λ_{C}, the simplest of which has been sketched in the preceding section. We base our extension on the work of Stefan Kaes, whose work on parametric overloading will be discussed later. Our two points of reference are [Kae88] and [Kae05], the latter being a vast elaboration and systematization of the former, and both of which present a partial solution to the problem of reconciling ad-hoc and parametric polymorphism by finding a common representation scheme. The benefit of this choice is an existing development of a denotational semantics for overloaded operators, together with a systematic approach to extending Milner's type inference by adaptation of the unification algorithm. We improve upon the dynamic semantics defined by Kaes through slight changes to the translation process, which allow us to decouple compilation from type checking, thereby showing that type inference is optional.

4.2.1 Type Language for Type System $\mathbb{T}^{\mathrm{T}}$

Because we will rely on monomorphic type expressions in the definition of overloaded operators, we prepone presentation of the complete type language for the system $\mathbb{T}^{\mathrm{T}}$ to be developed later, although we will make use of only $\mathbb{M}^{\mathrm{PC}}$ for the time being. The reader may want to skip the other definitions and return to them later.

$m \in \mathbb{M}^{\mathrm{PC}}, m ::= \texttt{int} \mid m_1 \to m_2$

$$\mathbb{C} \in \mathcal{C} := \{\top\} \cup \mathcal{P}_{fin}(\{c_n(\overline{'\mathtt{a_n}}) \,|\, c_n \in \mathsf{C} \wedge {'\mathtt{a_i}} \in \mathbb{V} \wedge (i \neq j \Rightarrow {'\mathtt{a_i}} \neq {'\mathtt{a_j}})\}),$$

where we require each $\mathbb{C}$ to be of finite depth, i.e. there are no infinite descending chains.

$'\mathtt{a}_{\mathbb{C}} \in \mathbb{V}$, we write just $'\mathtt{a}$ if no constraint is to be indicated.

$$\mathbf{cst}('a_{\mathbb{C}}) := \mathbb{C}$$

$$\tau \in \mathbb{M}^{\mathrm{T}}, \tau ::= \mathtt{int} \mid {'\mathtt{a}} \mid \tau_1 \to \tau_2$$

$$\sigma \in \mathbb{P}^{\mathrm{T}}, \sigma ::= \tau \mid \dot{\forall}{'\mathtt{b_1}} \ldots {'\mathtt{b_n}}.\tau \mid \ddot{\forall}{'\mathtt{b_1}} \ldots {'\mathtt{b_n}}.\tau \text{ with } \{{'\mathtt{b_1}}, \ldots, {'\mathtt{b_n}}\} \subseteq \mathbf{FV}(\tau)$$

$$\Gamma \in \mathbb{H}^{\mathrm{T}} := \mathbb{X} \to \mathbb{P}^{\mathrm{T}}$$

$$T \in \mathbb{T}^{\mathrm{T}} := \mathcal{P}(\mathbb{H}^{\mathrm{T}} \times \mathbb{M}^{\mathrm{T}})$$

We introduce explicit syntactic forms for type schemes which allow us to differentiate between the two forms of polymorphism we want to accommodate.

4.2.2 The Language λ_{K}

We extend the language λ_{C} by additional syntactic forms and interpretation rules which allow the language user to define overloadings on operators, by lifting the domain-based dispatch functionality found in the examples in section 4.1.2 to the language level. In the below definitions, π is for "polymorphic".

$\mathtt{x}, \mathtt{f}, \ldots \in \mathbb{X}$: program variables

$z \in \mathbb{Z}$: integers

$c \in \mathsf{C} = \{\mathtt{int}, (\to)\}$: type constructors, doubling as runtime tags

$$\omega \in \mathbb{M}^{\mathrm{T}}_{\$} := \{w_1 \to \ldots \to w_n \to w_r \,\big|\, n \geq 1 \wedge w_i \in \mathbb{M}^{\mathrm{PC}} \cup \{\$\} \wedge\ w_r = \$ \Rightarrow \exists i \in \{1, \ldots, n\} : w_i = \$\}$$

$e \in \mathbb{E}$: program expressions

$$e ::= \mathtt{x} \mid \lambda\mathtt{x}.e \mid e_1 e_2 \mid (e) \mid \mu\mathtt{f}.\lambda\mathtt{x}.e \mid \mathtt{let\ x} = e_1 \mathtt{\ in\ } e_2 \mid \pi\mathtt{f} :: \omega \mathtt{\ in\ } e \mid \pi\mathtt{f}\ c_1 \ldots c_n = e_1 \mathtt{\ in\ } e_2 \mid z \mid e_1 + e_2 \mid \mathtt{ifz\ } e_1 \mathtt{\ then\ } e_2 \mathtt{\ else\ } e_3$$

Although λ_{K}, like λ_{C}, is an extremely limited language, we will take license to make use of *obvious* extensions when illustrating functionality. For an example of an overloading definition, consider a dispatching negation operator `neg`, which *negates* both integers and booleans:

```
π neg :: $ → $ in
  π neg int = (λx.x*(-1)) in
    π neg bool = (λx.if x = true then false else true) in
      if neg false then neg 5 else 5
```

4.2.3 Translation Without Type Inference

A Naïve Semantics

For an interpretation of this extended language, we first give a name to the "almost everywhere wrong function" $\varepsilon_{\blacktriangleright\!\!\blacktriangleleft} := \Lambda u.\mathsf{if}\ u = \bot\ \mathsf{then}\ \bot\ \mathsf{else}\ \blacktriangleright\!\!\blacktriangleleft$. The predicate is-a establishes the expected correspondence between type constructors from C understood as names for runtime tags and their respective domains. For example, $\uparrow(5) :: \mathbb{Z}_\bot$ is-a `int`. We first give the idea for a naïve, though in principle reasonable, extension of λ_C.

$$
\begin{aligned}
\mathbf{S}[\![\pi\mathtt{f} :: \omega\ \mathtt{in}\ e]\!] &:= \Lambda R.\mathbf{S}[\![e]\!]R[\mathtt{f} \leftarrow \varepsilon_{\blacktriangleright\!\!\blacktriangleleft}] \\
\mathbf{S}[\![\pi\mathtt{f}\ c_1 \ldots c_n = e_1\ \mathtt{in}\ e_2]\!] &:= \Lambda R.\uparrow(\Lambda u_1 \ldots u_n. \\
&\qquad \mathsf{if}\ \bot \in \{u_1, \ldots, u_n\}\ \mathsf{then}\ \bot \\
&\qquad \mathsf{else\ if}\ \blacktriangleright\!\!\blacktriangleleft \in \{u_1, \ldots, u_n\}\ \mathsf{then}\ \blacktriangleright\!\!\blacktriangleleft \\
&\qquad \mathsf{else\ if}\ u_i\ \text{is-a}\ c_i\ \mathsf{then} \\
&\qquad\quad \mathbf{S}[\![e_1]\!]R(u_1) \ldots (u_n) \\
&\qquad \mathsf{else}\ \mathbf{S}[\![e_2]\!]R(u_1) \ldots (u_n)) :: [\mathbb{U} \to \mathbb{U}]_\bot
\end{aligned}
$$

However, this semantics forces language users to define overloadings in specific order that builds from primitive to composed values and generally reduces extensibility[4]. In a fashion similar to [Kae88, Kae05], but with more explicit information handling due to our type-agnostic compilation strategy, we therefore change the definition of a program environment to include unsatisfied identifiers for overloaded functions which require an operator environment for evaluation.

A Semantics With Indirection

Definition 4.2.1 (Overloaded Program Environments)**.**

$$
\begin{aligned}
O \in \mathbb{O} &\cong \mathbb{X} \to [\mathsf{C}] \to \mathbb{O} \to \mathbb{U} \\
R \in \mathbb{R}' &:= \mathbb{X} \to (\mathbb{U} \oplus (\mathbb{O} \to \mathbb{U})) \\
(R, O) \in \mathbb{Q} &:= \mathbb{R}' \times \mathbb{O}
\end{aligned}
$$

For convenience, we define a function that evaluates an identifier to its value in a given overloaded environment (R, O).

$$
\begin{aligned}
&\mathrm{eval} : \mathbb{R}' \times \mathbb{O} \to \mathbb{X} \to \mathbb{U} \\
&\mathrm{eval}_{R,O}(\mathtt{x}) := \mathsf{if}\ R(\mathtt{x}) \in \mathbb{O} \to \mathbb{U}\ \mathsf{then}\ R(\mathtt{x})(O)\ \mathsf{else}\ R(x)
\end{aligned}
$$

We may now give the semantics $\dot{\mathbb{S}} := \mathbb{S}_\mathbb{Q}$ for λ_K. We use [] to denote an empty finite map, $[v_1/x_1, v_2/x_2, \ldots, v_n/z_n]$ for a finite map, and write $F[\bar{v}/\bar{x}]$ for the composition $F \circ [\bar{v}/\bar{x}]$, where F need not be a finite map. We say that overloading assumption O admits arity n for identifier `f` ($\mathtt{f} \overset{n}{\Diamond} O$), if for all $\langle \bar{c} \rangle \in \mathrm{dom}(O(\mathtt{f}))$, $\langle \bar{c} \rangle$ has length n. We write $\dot{\mathbf{S}}[\![\cdot]\!]$ for $\mathbf{S}_\mathbb{Q}[\![\cdot]\!]$.

[4] A restriction that our type system *will* unfortunately force.

Other than the added cases for the two new syntactic forms, the only substantial changes occur in the definition of variable look-up. In the remaining cases the added parameter is merely passed through.

$$
\begin{array}{ll}
\dot{\mathbf{S}}[\![\mathtt{x}]\!] & := \Lambda R.\Lambda O.\mathrm{eval}_{R,O}(\mathtt{x}) \\
\dot{\mathbf{S}}[\![\lambda \mathtt{x}.e]\!] & := \Lambda R.\Lambda O.\uparrow(\Lambda u.\mathsf{if}\ u \in \{\bot, \blacktriangleright\!\!\blacktriangleleft\}\ \mathsf{then}\ u \\
 & \qquad\qquad \mathsf{else}\ \dot{\mathbf{S}}[\![e]\!]R[\mathtt{x} \leftarrow u]O) :: [\mathbb{U} \to \mathbb{U}]_\bot \\
\dot{\mathbf{S}}[\![e_1 e_2]\!] & := \Lambda R.\Lambda O.\mathsf{if}\ \bot \in \{\dot{\mathbf{S}}[\![e_1]\!]RO, \dot{\mathbf{S}}[\![e_2]\!]RO\}\ \mathsf{then}\ \bot \\
 & \qquad\qquad \mathsf{else\ if}\ \dot{\mathbf{S}}[\![e_1]\!]RO = f :: [\mathbb{U} \to \mathbb{U}]_\bot\ \mathsf{then} \\
 & \qquad\qquad\quad \downarrow(f)(\dot{\mathbf{S}}[\![e_2]\!]RO) \\
 & \qquad\qquad \mathsf{else}\ \blacktriangleright\!\!\blacktriangleleft \\
\dot{\mathbf{S}}[\![\mu \mathtt{x}.\lambda \mathtt{f}.e]\!] & := \Lambda R.\Lambda O.\mathbf{lfp}^{\sqsubseteq}_{\bar{\varepsilon}_\bot}(\Lambda\phi.\dot{\mathbf{S}}[\![\lambda \mathtt{x}.e]\!]R[\mathtt{f} \leftarrow \phi]O) \\
\dot{\mathbf{S}}[\![\mathtt{let\ x} = e_1\ \mathtt{in}\ e_2]\!] & := \Lambda R.\Lambda O.\dot{\mathbf{S}}[\![(\lambda x.e_2)e_1]\!]RO \\
\dot{\mathbf{S}}[\![\pi \mathtt{f} :: \omega\ \mathtt{in}\ e]\!] & := \Lambda R.\Lambda O.\dot{\mathbf{S}}[\![e]\!]R[\mathbf{resolve}_{\mathbf{arity}(\omega)}(\mathtt{f})/\mathtt{f}]O[[]/\mathtt{f}] \\
\dot{\mathbf{S}}[\![\pi \mathtt{f}\ c_1 \ldots c_n = e_1\ \mathtt{in}\ e_2]\!] & := \Lambda R.\Lambda O.\mathsf{if}\ \mathtt{f} \in \mathrm{dom}(O), R(\mathtt{f}) \in \mathbb{O} \to \mathbb{U}, \mathtt{f} \overset{n}{\widehat{\ominus}} O\ \mathsf{then} \\
 & \qquad\qquad \dot{\mathbf{S}}[\![e_2]\!]RO[O(\mathtt{f})[\dot{\mathbf{S}}[\![e_1]\!]R/\langle c_1, \ldots, c_n\rangle]/\mathtt{f}] \\
 & \qquad\qquad \mathsf{else}\ \blacktriangleright\!\!\blacktriangleleft \\
\dot{\mathbf{S}}[\![z]\!] & := \Lambda R.\Lambda O.\uparrow(z) :: \mathbb{Z}_\bot \\
\dot{\mathbf{S}}[\![e_1 + e_2]\!] & := \Lambda R.\Lambda O.\mathsf{if}\ \bot \in \{\dot{\mathbf{S}}[\![e_1]\!]RO, \dot{\mathbf{S}}[\![e_2]\!]RO\}\ \mathsf{then}\ \bot \\
 & \qquad\qquad \mathsf{else\ if}\ \dot{\mathbf{S}}[\![e_i]\!]RO = \uparrow(z_i) :: \mathbb{Z}_\bot\ \mathsf{then} \\
 & \qquad\qquad\quad \uparrow(z_1 + z_2) :: \mathbb{Z}_\bot \\
 & \qquad\qquad \mathsf{else}\ \blacktriangleright\!\!\blacktriangleleft \\
\dot{\mathbf{S}}[\![\mathtt{ifz}\ e_1\ \mathtt{then}\ e_2\ \mathtt{else}\ e_3]\!] & := \Lambda R.\Lambda O.\mathsf{if}\ \dot{\mathbf{S}}[\![e_1]\!]R = \bot\ \mathsf{then}\ \bot \\
 & \qquad\qquad \mathsf{else\ if}\ \dot{\mathbf{S}}[\![e_1]\!]RO = \uparrow(z) :: \mathbb{Z}_\bot\ \mathsf{then} \\
 & \qquad\qquad\quad \dot{\mathbf{S}}[\![\mathsf{if}\ z = 0\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3]\!]RO \\
 & \qquad\qquad \mathsf{else}\ \blacktriangleright\!\!\blacktriangleleft
\end{array}
$$

The function $\mathrm{resolve}_n : \mathbb{X} \to \mathbb{O} \to \mathbb{U}$ creates an n-ary dispatching function which may be added to the runtime environment on defining an overloaded operator, and which uses the operator environment at call site to determine the implementation to delegate to. Note that we require eager checking of the conditions imposed on the u_i, that is checks are performed as far as possible for partial applications of the dispatching function. We use a simplified notation, however, instead of spelling out the details of eager checking.

$$
\begin{array}{l}
\mathbf{resolve}_n(\mathtt{f}) = \Lambda O.\Lambda u_1 \ldots u_n.\mathsf{if}\ u_i = \bot\ \mathsf{then}\ \bot \\
\qquad\qquad\qquad\qquad \mathsf{else\ if}\ \mathtt{f} \in \mathrm{dom}(O), \\
\qquad\qquad\qquad\qquad\quad \langle c_1, \ldots, c_n\rangle \in \mathrm{dom}(O(\mathtt{f})), \\
\qquad\qquad\qquad\qquad\quad \forall i = 1 \ldots n : u_i\ \text{is-a}\ c_i
\end{array}
$$

$$\mathsf{then}\ O(\mathtt{f})(\langle c_1, \ldots, c_n\rangle)(O)(u_1)\ldots(u_n)$$
$$\mathsf{else}\ \text{✘}$$

Note that the condition in the above definition varies slightly from that in [Kae05] (which already improves upon that in [Kae88])—instead of dispatching only on the overloaded parameters, we dispatch on *all*. This is to adjust for the fact that we have no compile-time notion of type, and hence have to process this information at runtime.

4.3 A Type System for λ_{K}

The type system $\mathbb{T}^{\mathrm{T}}$ is an extension of [Mil78, DM82] and vast simplification of the approach of [Kae88]. As in the work of Kaes, type variables are annotated with constraints, however the association with types is direct and local. This simplifies the type system, but also reduces its power as will be discussed below. The first and fundamental step is to adapt unification to work with constrained variables.

4.3.1 Unification with Constrained Variables

Substitutions

In different contexts we will require substitutions with either ground or variable-containing targets. We therefore define a general notion of substitution which we will instantiate as needed, usually assuming the nature of the instantiation to be obvious from the context.

Definition 4.3.1 (Substitution)**.** A *T-substitution* is a total map $S : \mathbb{V} \to T$ assigning terms from T to type variables. Substitutions are lifted to type terms and type schemes in the usual way, such that bound variables in schemes remain untouched but are renamed as required to prevent capture.

We write $[{}'\mathtt{a}_1/t_1, \ldots, {}'\mathtt{a}_n/t_n]$ for the substitution assigning t_i to ${}'\mathtt{a}_i$ and acting as the identity on all other type variables (we therefore write [] for the empty, or identity, substitution). Application of substitutions is written either in prefix or functional form. Composition of substitutions is defined in the usual way.

For example, we write $S : \mathbb{V} \to \mathbb{M}^{\mathrm{T}}, S' : \mathbb{V} \to \mathbb{P}^{\mathrm{T}}$ to identify S and S' as $\mathbb{M}^{\mathrm{T}}$- and $\mathbb{P}^{\mathrm{T}}$-substitutions, respectively.

Definition 4.3.2 (Respectful Substitution)**.** A substitution S is *respectful* if for all ${}'\mathtt{a} \in \mathrm{dom}(S)$ we have

$$S({}'\mathtt{a}) <: \mathbf{cst}({}'\mathtt{a}),$$

where

$$\begin{aligned}
\tau <: \top \quad & \text{for all } \tau, \\
{}'\mathtt{a}_Y <: X \quad & \text{if } \forall c(\overline{{}'\mathtt{a}_n}) \in Y : \exists \overline{{}'\mathtt{b}_n} : \big(c(\overline{{}'\mathtt{b}_n}) \in X \wedge {}'\mathtt{a}_i <: \mathbf{cst}({}'\mathtt{b}_i)\big), \\
c(\overline{\tau_n}) <: X \quad & \text{if } \exists \overline{{}'\mathtt{a}_n} : \big(c(\overline{{}'\mathtt{a}_n}) \in X \wedge \tau_i <: \mathbf{cst}({}'\mathtt{a}_i)\big).
\end{aligned}$$

We write $S : \mathbb{V} \xrightarrow{r} T$ to indicate that S is a respectful T-substitution.

We use the notion of respectful substitutions to define an ordering on type terms in the usual way, where we call τ_1 a *valid instance* of τ_2 ($\tau_1 \leq^{\mathrm{T}} \tau_2$) if there is a respectful substitution S such that $\tau_1 = S(\tau_2)$. This ordering is extended to respectful substitutions pointwise: We say that S_2 is more general than S_1 ($S_1 \leq^{\mathrm{T}} S_2$), if $\mathrm{dom}(S_1) \subseteq \mathrm{dom}(S_2)$ and, for all $'\mathtt{a} \in \mathrm{dom}(S_1), S_1('\mathtt{a}) \leq^{\mathrm{T}} S_2('\mathtt{a})$.

Proposition 4.3.3. *For any variable constraint $\mathbb{C}$, type term τ and respectful substitution S,*

$$\tau <: \mathbb{C} \Rightarrow S(\tau) <: \mathbb{C}.$$

Proof. By induction on the structure of τ, using the fact that if $'\mathtt{b} <: \mathbf{cst}('\mathtt{a})$ and $'\mathtt{c} <: \mathbf{cst}('\mathtt{b})$, then $'\mathtt{c} <: \mathbf{cst}('\mathtt{a})$. □

As an immediate consequence we get:

Corollary 4.3.4 (Respect Composes)**.** *If S and S' are respectful substitutions, then $S \circ S'$ is respectful.*

Constrained Terms and Unification

The unification method required for terms with constrained variables, as well as its proof of adequacy, is modeled on that presented in [Kae05, p. 54], which in turn is a variant of Robinson's algorithm (see for example [BS01] for a survey). The main difference is that in the variable case we can not simply unify the variable with any type term, but need to respect the variable's constraints.

To be able to do so, we define a function $\mathbf{constrain} : \mathcal{C} \times \mathbb{M}^{\mathrm{T}} \to (\mathbb{V} \to \mathbb{M}^{\mathrm{T}})$ which from a variable constraint and a type term produces a substitution that can be used to adjust the type term to match the constraints. The process fails if the type term is incompatible with the constraints. In the below definition, $'\mathtt{b}$ is a fresh variable.

$$
\begin{array}{lll}
\mathbf{constrain}(\top, \tau) & := [] & \\
\mathbf{constrain}(X, '\mathtt{a}_\top) & := ['\mathtt{b}_X / '\mathtt{a}_\top] & \\
\mathbf{constrain}(X, '\mathtt{a}_Y) & := ['\mathtt{b}_Z / '\mathtt{a}_Y] & \text{if } Z = \big\{ c(S_1('\mathtt{c_1}), \ldots, S_n('\mathtt{c_n})) \,\big|\, \\
 & & \quad c(\overline{'\mathtt{c_n}}) \in Y \wedge c(\overline{'\mathtt{d_n}}) \in X \\
 & & \quad \wedge \forall i = 1, \ldots, n : S_i = \mathbf{constrain}(\mathbf{cst}('\mathtt{d_i}), '\mathtt{c_i}) \\
 & & \big\} \neq \emptyset \\
\mathbf{constrain}(X, c(\overline{'\tau_\mathtt{n}})) & := S_n & \text{if } c(\overline{'\mathtt{c_n}}) \in X \wedge \exists S_0 \ldots S_{n-1} : \big(S_0 = [] \\
 & & \quad \wedge \forall i = 1, \ldots, n : \\
 & & \quad S_i = \mathbf{constrain}(\mathbf{cst}('\mathtt{c_i}), S_{i-1}(\tau_i)) \circ S_{i-1}\big)
\end{array}
$$

Proposition 4.3.5. *For a variable constraint $\mathbb{C}$ and a type term τ, if there is a respectful substitution for which $S(\tau) <: \mathbb{C}$, then there is a most general one. We may compute the most general substitution as* $\mathbf{constrain}(\mathbb{C}, \tau)$.

Proof. We proceed in two phases, showing first that if $S = \mathbf{constrain}(\mathbb{C}, \tau)$, then S is respectful and $S(\tau) <: \mathbb{C}$. The case of $\mathbb{C} = \top$ is obvious.

For $\mathbb{C} = X$, we perform an induction on the structure of type term τ. Because the base of our induction on τ contains a variable constraint as another inductive structure, we must first consider the cases where the variable constraint is either $\top$ or a finite set of nullary type constructors. In both cases it is easy to see that **constrain** produces the desired result. Let then $\tau = {'\mathtt{a}}_Y$, we have $\mathbf{constrain}(X, {'\mathtt{a}}_Y)({'\mathtt{a}}_Y) = {'\mathtt{b}}_Z$ with $'\mathtt{b}$ fresh. By the definition of Z and our subinduction hypothesis, we may conclude ${'\mathtt{b}}_Z <: X$.

Then if $\tau = c(\overline{\tau_n})$, we know $c(\overline{'\mathtt{c_n}}) \in X$. Each of the S_i is respectful, as we may conclude from the fact that the identity S_0 is respectful, by repeated use of our induction hypothesis and 4.3.4. As we may factor S_n as $S'_i \circ S_i$ for each i, $S_n(c(\tau_1, \ldots, \tau_n)) = c(S'_1 \circ S_1(\tau_1), \ldots, S'_n \circ S_n(\tau_n)) <: X$, as is established by n applications of induction hypothesis and 4.3.3.

We now need to establish that $S = \mathbf{constrain}(\mathbb{C}, \tau)$ is indeed most general, that is, that for each respectful R for which $R(\tau) <: \mathbb{C}$, there is respectful S' such that $R = S' \circ S$. We again proceed by induction on τ. If $\tau = {'\mathtt{a}}$, choose $S' = R[R('\mathtt{a})/S('\mathtt{a})]$, then $S'(S('\mathtt{a})) = R(a)$, while all other variables are unaffected and hence $S' \circ S = R$.

For $\tau = c(\tau_1, \ldots, \tau_n)$ we perform a subinduction on $i = 0, \ldots, n$ to show that in each case we can factor R as $S'_i \circ S_i$. For $i = 0$, the fact that $S_0 = []$ forces $S'_0 = R$. We may now assume $R = S'_{i-1} \circ S_{i-1}$. By outer induction hypothesis we may factor S'_{i-1} as $R' \circ \mathbf{constrain}(\mathbf{cst}('\mathtt{c_i}), S_{i-1}(\tau_i))$, but then $R = S'_{i-1} \circ S_{i-1} = (R' \circ \mathbf{constrain}(\mathbf{cst}('\mathtt{c_i}), S_{i-1}(\tau_i))) \circ S_{i-1} = R' \circ S_i$, so R' is the required S'_i. □

This property of **constrain** makes it appropriate for use in the definition of a unification procedure.

Definition 4.3.6 (Unification)**.**

$$
\begin{array}{lll}
\mathcal{U}(\tau, \tau) & := [] & \\
\mathcal{U}(\tau, '\mathtt{a}) & := \mathcal{U}('\mathtt{a}, \tau) & \text{unless } \tau \in \mathbb{V} \\
\mathcal{U}('\mathtt{a}, \tau) & := [S(\tau)/'\mathtt{a}] \circ S & \text{if } '\mathtt{a} \notin \mathbf{FV}(\tau), S = \mathbf{constrain}(\mathbf{cst}('\mathtt{a}), \tau) \\
\mathcal{U}(c(\overline{\tau_n}), c(\overline{\tau'_n})) & := S_n & \text{if } \exists S_0, \ldots, S_{n-1} : \big(S_0 = [] \wedge \forall i = 1, \ldots, n : \\
 & & \quad S_i = \mathcal{U}(S_{i-1}(\tau_i), S_{i-1}(\tau'_i)) \circ S_{i-1}\big)
\end{array}
$$

Theorem 4.3.7. *For $\tau_1, \tau_2 \in \mathbb{M}^{\mathrm{T}}$, if there is a respectful substitution S that unifies both terms—that is $S(\tau_1) = S(\tau_2)$—, then there is a most general such substitution. It can be computed as $\mathcal{U}(\tau_1, \tau_2)$.*

Proof. The proof is similar in structure to the proof of Proposition 4.3.5, which we rely on in the variable case $\mathcal{U}('\mathtt{a}, \tau)$ to argue that we can find a respectful substitution if and only if τ can be constrained to match $\mathbf{cst}('\mathtt{a})$ and $'\mathtt{a}$ does not occur in τ. □

4.3.2 A Rule-based Definition of $\mathbb{T}^{\mathrm{T}}$

We now give a rule-based type system for this language, which is heavily inspired by the systems presented in [Kae88, Kae05]. The reader may want to refer back to the definitions of 4.2.1 at this point.

Definition 4.3.8.

$$\frac{\tau \leq^{\mathrm{T}} \Gamma(\mathtt{x})}{\Gamma \vdash \mathtt{x} : \tau} \quad (\mathsf{Var})$$

$$\frac{\Gamma, \mathtt{x} : \tau \vdash e : \tau'}{\Gamma \vdash \lambda \mathtt{x}.e : \tau \to \tau'} \quad (\mathsf{Abs}) \qquad \frac{\Gamma \vdash e_1 : \tau \to \tau' \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash e_1 e_2 : \tau'} \quad (\mathsf{App})$$

$$\frac{\Gamma, \mathtt{f} : \tau \to \tau' \vdash \lambda \mathtt{x}.e : \tau \to \tau'}{\Gamma \vdash \mu \mathtt{f}.\lambda \mathtt{x}.e : \tau \to \tau'} \quad (\mathsf{Fix})$$

$$\frac{\Gamma \vdash e_1 : \tau \qquad \Gamma, x : \mathbf{gen}(\Gamma, \tau) \vdash e_2 : \tau'}{\Gamma \vdash \mathtt{let\ x\ =}\ e_1\ \mathtt{in}\ e_2 : \tau'} \quad (\mathsf{Let})$$

where $\mathbf{gen}(\Gamma, \tau) := \dot{\forall}\overline{{}'\mathtt{a_n}}.\tau$ for $\{{}'\mathtt{a}_1, \ldots, {}'\mathtt{a_n}\} = \mathbf{FV}(\tau) \setminus \mathbf{FV}(\Gamma)$

$$\frac{\Gamma \vdash e_1 : \mathtt{int} \qquad \Gamma \vdash e_2 : \tau \qquad \Gamma \vdash e_3 : \tau}{\Gamma \vdash \mathtt{ifz}\ e_1\ \mathtt{then}\ e_2\ \mathtt{else}\ e_3 : \tau} \quad (\mathsf{Ifz})$$

$$\frac{\Gamma, \mathtt{f} : \ddot{\forall}'\mathtt{a}_\emptyset.[{}'\mathtt{a}_\emptyset/\$]\omega \vdash e : \tau}{\Pi; \Gamma \vdash \pi \mathtt{f} :: \omega\ \mathtt{in}\ e : \tau} \quad (\mathsf{Op\text{-}Intro})$$

$$\frac{\begin{array}{c}\Gamma, \mathtt{f} : \ddot{\forall}'\mathtt{a}_X.\tau_0 \vdash e_1 : \tau_1 \qquad \Gamma, \mathtt{f} : \ddot{\forall}'\mathtt{a}_{X \cup \{c(\overline{{}'\mathtt{c}})\}}.[{}'\mathtt{a}_{X \cup \{c(\overline{{}'\mathtt{c}})\}}/{}'\mathtt{a}_X]\tau_0 \vdash e_2 : \tau_2 \\ \tau_1 \leq^{\mathrm{T}} c_1(\overline{{}'\mathtt{a}^{\mathtt{1}}_\top}) \to \ldots \to c_n(\overline{{}'\mathtt{a}^{\mathtt{n}}_\top}) \to {}'\mathtt{b}_\top \qquad \tau_1 = [c(\overline{{}'\mathtt{c}})/{}'\mathtt{a}_X]\omega \qquad c \notin d\end{array}}{\Gamma, \mathtt{f} : \ddot{\forall}'\mathtt{a}_X.\tau_0 \vdash \pi \mathtt{f}\ c_1 \ldots c_n = e_1\ \mathtt{in}\ e_2 : \tau_2} \quad (\mathsf{Op\text{-}Def})$$

The remaining rules for the base types are less crucial to the principles of the system, but given for completeness:

$$\frac{}{\Gamma \vdash z : \mathtt{int}} \quad (\mathsf{Int\text{-}Base}) \qquad \frac{\Gamma \vdash e_1 : \mathtt{int} \qquad \Gamma \vdash e_2 : \mathtt{int}}{\Gamma \vdash e_1 + e_2 : \mathtt{int}} \quad (\mathsf{Int\text{-}Plus})$$

The type inference algorithm for this system is a slight modification of the algorithm $\mathcal{W}$ [DM82], with constrained unification substituted for Robinson unification.

Chapter 5

Abstract Interpretation of Overloading

It's turtles all the way *up!*
—Steve Awodey

We now want to obtain a soundness result for the type system presented above in definition 4.3.8, so as to ensure that "well-typed expressions do not go wrong". Our strategy is to employ the abstract interpretation framework and constructions introduced in Chapter 3, where we first need to establish that equivalent results hold for our extended language λ_{K}. We then build on said results to establish the soundness of the new type system by proving it to be a sound abstraction.

5.1 Type Collecting Semantics for Programs with Overloading

In order to update the type collecting semantics, we need not change the definition of collecting polytypes given in 3.3.2, as it is expressive enough for our purposes. What needs to be updated are the connections to expressions, where new syntactic forms have been introduced, and to program properties, which now are elements of $\mathbb{P}_{\mathbb{Q}}$ rather than $\mathbb{P}_{\mathbb{R}}$.

In order to do so, we first consider the interpretation of type environments $H \in \mathbb{H}^{\mathrm{Co}}$, which now need to relate to overloaded program environments $(R, O) \in \mathbb{Q}$. We therefore redefine the abstraction function for environments to evaluate identifiers in overloaded instead of regular environments

$$\alpha_2^{\mathrm{Co}}(E) := \Lambda \mathtt{x}.\alpha_1^{\mathrm{Co}}(\{\mathrm{eval}_{R,O}(\mathtt{x}) \mid (R, O) \in E\}),$$

and the concretization function accordingly

$$\gamma_2^{\mathrm{Co}}(H) := \{(R, O) \in \mathbb{R}' \times \mathbb{O} \mid \forall \mathtt{x} \in \mathbb{X} : \mathrm{eval}_{R,O}(\mathtt{x}) \in \gamma_1^{\mathrm{Co}}(H(x))\}.$$

The changes propagate in a straightforward way to the definitions of the corresponding functions for typings, where $\theta \in \mathbb{T}^{\mathrm{Co}} := \mathbb{H}^{\mathrm{Co}} \to \mathbb{P}^{\mathrm{Co}}_{\equiv}$,

$$\alpha^{\mathrm{Co}}(P) := \Lambda H.\alpha_1^{\mathrm{Co}}(\{\phi(R)(O) \mid (R,O) \in \gamma_2^{\mathrm{Co}}(H) \wedge \phi \in P\})$$

$$\gamma^{\mathrm{Co}}(\theta) := \{\phi \mid \forall H \in \mathbb{H}^{\mathrm{Co}} : \forall (R,O) \in \gamma_2^{\mathrm{Co}}(H) : \phi(R)(O) \in \gamma_1^{\mathrm{Co}}(\theta(H))\}$$

We furthermore need to provide definitions of the semantic function for the added syntactic forms, for which we first introduce a shorthand for constructing a polymorphic function type, which we use to adjust for the membership checks used in dynamic dispatch.

Definition 5.1.1 (Polymorphic Guarded Function Types)**.** For a finite sequence of constructor symbols $s = c_1, \ldots, c_n$, define the polytype of s-guarded functions as

$$\mathbb{P}^{Co}_{\vec{n}}(c_1, \ldots, c_n) := \{c_1(\bar{t_1}) \to \ldots \to c_n(\bar{t_n}) \to \bot^{\mathrm{Co}} \mid \bar{t_1}, \ldots, \bar{t_n} \in \mathbb{P}^{\mathrm{Co}}_{\equiv}\},$$

such that the arity of each c_i is respected.

For the introduction of new overloadings on identifiers, we merely associate the identifier with the empty type of $\mathbb{P}^{\mathrm{Co}}_{\equiv}$, thereby ensuring that applications of this operator can not be typed. The definition for overloading definitions is more intricate. We need to combine the polytype of the pre-existing implementations with that of the one being defined, but only in case the actual type of the implementation has the right arity and conforms to the dynamic dispatch cues provided by the $c_1, \ldots, c_n$.

$$\begin{aligned}
&\mathbf{T}^{\mathrm{Co}}[\![\pi \mathtt{f} :: \omega \ \mathtt{in}\ e]\!]H := \\
&\quad \mathbf{T}^{\mathrm{Co}}[\![e]\!]H[\mathtt{f} : \emptyset^{\mathrm{Co}}] \\
&\mathbf{T}^{\mathrm{Co}}[\![\pi \mathtt{f}\ c_1 \ldots c_n = e_1\ \mathtt{in}\ e_2]\!]H := \\
&\quad \bigwedge^{Co} \left\{ \mathbf{T}^{\mathrm{Co}}[\![e_2]\!]H[\mathtt{f} : H(\mathtt{f}) \wedge^{Co} T] \,\middle|\, T = \mathbf{T}^{\mathrm{Co}}[\![e_1]\!]H \vee^{Co} \mathbb{P}^{Co}_{\vec{n}}(c_1, \ldots, c_n) \wedge \right. \\
&\qquad \left. H(\mathtt{f}), T \leq^{Co} \{\bot^{\mathrm{Co}} \to \ldots \to \bot^{\mathrm{Co}} \to \emptyset^{\mathrm{Co}}\} \right\}
\end{aligned}$$

We need an adapted version of Proposition 3.3.8, which in re-stated form is:

Proposition 5.1.2.

$$\begin{aligned}
&\forall e \in \mathbb{E} : \forall H \in \mathbb{H}^{\mathrm{Co}} : \forall t \in \mathbb{P}^{\mathrm{Co}} : \\
&\quad \alpha^{\mathrm{Co}}(\mathbf{C}[\![e]\!])H \leq^{Co} t \Leftrightarrow \forall (R,O) \in \gamma_2^{\mathrm{Co}}(H) : \dot{\mathbf{S}}[\![e]\!]RO \in \gamma_1^{\mathrm{Co}}(t)
\end{aligned}$$

Proof. This is easily checked. □

Proposition 3.3.10 requires checking of all the old and both of the new cases. We show the base case along with those for the new syntactic forms, as all other cases translate naturally (note that those are the only cases in which the overloading assumption is directly used in the semantic function for λ_{K}). The

interested reader is again referred to either or both of [Cou97, p. 325] and A.1. We begin with the variable case.

$$
\begin{array}{lll}
 & \alpha^{\mathrm{Co}}(\mathbf{C}[\![\mathtt{x}]\!])H & \\
= & \bigwedge^{Co}\left\{t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \mid \forall (R,O) \in \gamma_2^{\mathrm{Co}}(H) : \dot{\mathbf{S}}[\![\mathtt{x}]\!]RO \in \gamma_1^{\mathrm{Co}}(t)\right\} & \\
= & \bigwedge^{Co}\left\{t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \mid \forall (R,O) \in \mathbb{R}' \times \mathbb{O} : \right. & \\
 & \quad \left.\left(\forall \mathtt{y} \in \mathbb{X} : \mathrm{eval}_{R,O}(\mathtt{y}) \in \gamma_1^{\mathrm{Co}}(H(\mathtt{y})) \Rightarrow \mathrm{eval}_{R,O}(\mathtt{x}) \in \gamma_1^{\mathrm{Co}}(t)\right)\right\} & \\
\leq^{\mathrm{Co}} & \bigwedge^{Co}\left\{t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \mid \forall (R,O) \in \mathbb{R}' \times \mathbb{O} : \right. & \\
 & \quad \left.\left(\mathrm{eval}_{R,O}(\mathtt{x}) \in \gamma_1^{\mathrm{Co}}(H(\mathtt{x})) \Rightarrow \mathrm{eval}_{R,O}(\mathtt{x}) \in \gamma_1^{\mathrm{Co}}(t)\right)\right\} & \\
= & \bigwedge^{Co}\left\{t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \mid \forall u \in \mathbb{U} : \left(u \in \gamma_1^{\mathrm{Co}}(H(\mathtt{x})) \Rightarrow u \in \gamma_1^{\mathrm{Co}}(t)\right)\right\} & \\
= & \bigwedge^{Co}\left\{t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \mid \gamma_1^{\mathrm{Co}}(H(\mathtt{x})) \subseteq \gamma_1^{\mathrm{Co}}(t)\right\} & \\
= & \bigwedge^{Co}\left\{t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \mid H(\mathtt{x}) \leq^{Co} t\right\} & \\
= & H(\mathtt{x}) & \\
= & \mathbf{T}^{\mathrm{Co}}[\![\mathtt{x}]\!]H &
\end{array}
$$

We state an easily verified fact about infima that we used in the variable case and which will be of repeated use in upcoming proofs.

Proposition 5.1.3. *For any predicates ϕ, ψ, if for all $t \in \mathbb{P}^{\mathrm{Co}}_{\equiv}$ we have $\phi(t) \Rightarrow \psi(t)$, then $\bigwedge^{Co}\{t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \mid \psi(t)\} \leq^{Co} \bigwedge^{Co}\{t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \mid \phi(t)\}$.*

For the introduction of overloaded operators, the important observation is that the dispatch function added to the environment conforms to a function type scheme, applications of which can not be typed.

$$
\begin{array}{lll}
 & \alpha^{\mathrm{Co}}(\mathbf{C}[\![\pi\mathtt{f} :: \omega \ \mathtt{in}\ e]\!])H & \\
= & \bigwedge^{Co}\left\{t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \mid \forall (R,O) \in \gamma_2^{\mathrm{Co}}(H) : \dot{\mathbf{S}}[\![\pi\mathtt{f} :: \omega \ \mathtt{in}\ e]\!]RO \in \gamma_1^{\mathrm{Co}}(t)\right\} & \\
= & \bigwedge^{Co}\left\{t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \mid \forall (R,O) \in \gamma_2^{\mathrm{Co}}(H) : \right. & \\
 & \quad \left.\dot{\mathbf{S}}[\![e]\!]R[\mathbf{resolve}_{\mathbf{arity}(\omega)}(\mathtt{f})/\mathtt{f}]O[[]/\mathtt{f}] \in \gamma_1^{\mathrm{Co}}(t)\right\} & \\
= & \bigwedge^{Co}\left\{t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \mid \forall (R,O) \in \mathbb{R}' \times \mathbb{O} : \right. & \text{def. } \gamma_2^{\mathrm{Co}}(\cdot) \\
 & \quad \Big(\forall \mathtt{x} \in \mathbb{X} : \mathrm{eval}_{R,O}(\mathtt{x}) \in \gamma_1^{\mathrm{Co}}(H(\mathtt{x})) & \\
 & \qquad \Rightarrow \dot{\mathbf{S}}[\![e]\!]R[\mathbf{resolve}_{\mathbf{arity}(\omega)}(\mathtt{f})/\mathtt{f}]O[[]/\mathtt{f}] \in \gamma_1^{\mathrm{Co}}(t)\Big)\Big\} & \\
\leq^{Co} & \bigwedge^{Co}\left\{t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \mid \forall (R,O) \in \mathbb{R}' \times \mathbb{O} : \right. & (5.1.3) \\
 & \quad \Big(\forall \mathtt{x} \in \mathbb{X} \setminus \{\mathtt{f}\} : \mathrm{eval}_{R,O}(\mathtt{x}) \in \gamma_1^{\mathrm{Co}}(H(\mathtt{x})) & \\
 & \qquad \Rightarrow \dot{\mathbf{S}}[\![e]\!]R[\mathbf{resolve}_{\mathbf{arity}(\omega)}(\mathtt{f})/\mathtt{f}]O[[]/\mathtt{f}] \in \gamma_1^{\mathrm{Co}}(t)\Big)\Big\} &
\end{array}
$$

Here we use the fact that for all (R, O)

$$\mathrm{eval}_{R[\mathbf{resolve}_{\mathbf{arity}(\omega)}(\mathtt{f})/\mathtt{f}],O[[]/\mathtt{f}]}(\mathtt{f}) = (\lambda u_1 \ldots u_{\mathbf{arity}(\omega)}.\cdots) \in \gamma_1^{\mathrm{Co}}(\emptyset^{\mathrm{Co}}),$$

and that, if $\mathtt{x} \neq \mathtt{f}$,

$$\mathrm{eval}_{R[\mathbf{resolve}_{\mathbf{arity}(\omega)}(\mathtt{f})/\mathtt{f}],O[[]/\mathtt{f}]}(\mathtt{x}) = \mathrm{eval}_{R,O}(\mathtt{x}).$$

$$\begin{array}{lll}
\leq^{Co} & \bigwedge^{Co} \big\{ t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \mid \forall (R,O) \in \mathbb{R}' \times \mathbb{O} : & \\
& \quad \big(\forall \mathtt{x} \in \mathbb{X} \setminus \{\mathtt{f}\} : \mathrm{eval}_{R,O}(\mathtt{x}) \in \gamma_1^{\mathrm{Co}}(H(\mathtt{x})) & \\
& \qquad \Rightarrow \mathrm{eval}_{R,O}(\mathtt{f}) \in \gamma_1^{\mathrm{Co}}(\emptyset^{\mathrm{Co}}) \Rightarrow \dot{\mathbf{S}}[\![e]\!] RO \in \gamma_1^{\mathrm{Co}}(t) \big) \big\} & (5.1.3) \\
= & \bigwedge^{Co} \big\{ t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \mid \forall (R,O) \in \mathbb{R}' \times \mathbb{O} : \forall \mathtt{x} \in \mathbb{X} : & \\
& \quad \big(\mathrm{eval}_{R,O}(\mathtt{x}) \in \gamma_1^{\mathrm{Co}}(H[\mathtt{f} : \emptyset^{\mathrm{Co}}]) \Rightarrow \dot{\mathbf{S}}[\![e]\!] RO \in \gamma_1^{\mathrm{Co}}(t) \big) \big\} & \\
= & \bigwedge^{Co} \big\{ t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \mid \forall (R,O) \in \gamma_2^{\mathrm{Co}}(H[\mathtt{f} : \emptyset^{\mathrm{Co}}]) : \dot{\mathbf{S}}[\![e]\!] RO \in \gamma_1^{\mathrm{Co}}(t) \big\} & \text{def. } \gamma_2^{\mathrm{Co}}(\cdot) \\
= & \bigwedge^{Co} \big\{ t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \mid \alpha^{\mathrm{Co}}(\mathbf{C}[\![e]\!]) H[\mathtt{f} : \emptyset^{\mathrm{Co}}] \leq^{Co} t \big\} & (5.1.2) \\
= & \alpha^{\mathrm{Co}}(\mathbf{C}[\![e]\!]) H[\mathtt{f} : \emptyset^{\mathrm{Co}}] & \text{lattice laws} \\
\leq^{\mathrm{Co}} & \mathbf{T}^{\mathrm{Co}}[\![e]\!] H[\mathtt{f} : \emptyset^{\mathrm{Co}}] & \text{by I.H.} \\
= & \mathbf{T}^{\mathrm{Co}}[\![\pi \mathtt{f} :: \omega \ \mathtt{in} \ e]\!] H & \text{by def.}
\end{array}$$

Finally, the case for overloading implementations.

$$\begin{array}{lll}
& \alpha^{\mathrm{Co}}(\mathbf{C}[\![\pi \mathtt{f} \ c_1 \ldots c_n = e_1 \ \mathtt{in} \ e_2]\!]) H & \\
= & \bigwedge^{Co} \big\{ t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \mid \forall (R,O) \in \gamma_2^{\mathrm{Co}}(H) : & \text{def. } \mathbf{C}[\![\cdot]\!] \\
& \quad \dot{\mathbf{S}}[\![\pi \mathtt{f} \ c_1 \ldots c_n = e_1 \ \mathtt{in} \ e_2]\!] RO \in \gamma_1^{\mathrm{Co}}(t) \big\} & \\
= & \bigwedge^{Co} \big\{ t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \mid \forall (R,O) \in \gamma_2^{\mathrm{Co}}(H) : \big(& \text{def. } \dot{\mathbf{S}}[\![\cdot]\!] \\
& \quad \mathtt{f} \in \mathrm{dom}(O) \wedge R(\mathtt{f}) \in \mathbb{O} \to \mathbb{U} \wedge \mathtt{f} \overset{n}{\widehat{\sim}} O & \\
& \qquad \Rightarrow \dot{\mathbf{S}}[\![e_2]\!] RO[O(\mathtt{f})[\dot{\mathbf{S}}[\![e_1]\!] R / \langle c_1, \ldots, c_n \rangle] / \mathtt{f}] \in \gamma_1^{\mathrm{Co}}(t) & \\
& \quad \mathtt{f} \notin \mathrm{dom}(O) \vee R(\mathtt{f}) \notin \mathbb{O} \to \mathbb{U} \vee \mathtt{f} \overset{n}{\not\sim} O & \\
& \qquad \Rightarrow \boldsymbol{\times} \in \gamma_1^{\mathrm{Co}}(t) \big) \big\} & \\
= & \bigwedge^{Co} \big\{ t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \mid \forall (R,O) \in \gamma_2^{\mathrm{Co}}(H) : \big(& \emptyset^{\mathrm{Co}} \text{ absorption} \\
& \quad \mathtt{f} \in \mathrm{dom}(O) \wedge R(\mathtt{f}) \in \mathbb{O} \to \mathbb{U} \wedge \mathtt{f} \overset{n}{\widehat{\sim}} O & \\
& \qquad \Rightarrow \dot{\mathbf{S}}[\![e_2]\!] RO[O(\mathtt{f})[\dot{\mathbf{S}}[\![e_1]\!] R / \langle c_1, \ldots, c_n \rangle] / \mathtt{f}] \in \gamma_1^{\mathrm{Co}}(t) \big) \big\} &
\end{array}$$

For this next step, we use the fact that for any $(R,O) \in \gamma_2^{\mathrm{Co}}(H)$, if $\mathtt{f} \overset{n}{\widehat{\sim}} O, R(\mathtt{f}) \in \mathbb{O} \to \mathbb{U}$ and $\mathtt{f} \in \mathrm{dom}(O)$, we have

$$\mathrm{eval}_{R,O[O(\mathtt{f})[\dot{\mathbf{S}}[\![e_1]\!] R / \langle c_1, \ldots, c_n \rangle / \mathtt{f}]}(\mathtt{f}) \in \gamma_1^{\mathrm{Co}}(\mathbf{T}^{\mathrm{Co}}[\![e_1]\!] H \vee^{Co} \mathbb{P}^{Co}_{\vec{n}}(c_1, \ldots, c_n))$$

as can be verified by partial evaluation and inspection of the resulting function, using the induction hypothesis for e_1. Hence, by instantiating (R,O) to the more specific form, we may strengthen to

$$\begin{array}{ll}
\leq^{Co} & \bigwedge^{Co} \big\{ t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \mid \forall (R,O) \in \gamma_2^{\mathrm{Co}}(H) : \\
& \quad \big(\mathrm{eval}_{R,O}(\mathtt{f}) \in \gamma_1^{\mathrm{Co}}(\mathbf{T}^{\mathrm{Co}}[\![e_1]\!] H \vee^{Co} \mathbb{P}^{Co}_{\vec{n}}(c_1, \ldots, c_n))
\end{array}$$

$$
\begin{array}{lll}
 & \quad \Rightarrow \dot{\mathbf{S}}[\![e_2]\!]RO \in \gamma_1^{\mathrm{Co}}(t)\big)\big\} & \\
= & \bigwedge^{Co} \big\{t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \mid \forall (R,O) \in \mathbb{R}' \times \mathbb{O} : & \text{def. } \forall, \gamma_2^{\mathrm{Co}}(\cdot) \\
 & \quad \big(\forall \mathtt{x} \in \mathbb{X} \setminus \{\mathtt{f}\} : \mathrm{eval}_{R,O}(\mathtt{x}) \in \gamma_1^{\mathrm{Co}}(H(\mathtt{x})) & \\
 & \quad \wedge\, \mathrm{eval}_{R,O}(\mathtt{f}) \in \gamma_1^{\mathrm{Co}}(H(\mathtt{f})) & \\
 & \quad \wedge\, \mathrm{eval}_{R,O}(\mathtt{f}) \in \gamma_1^{\mathrm{Co}}(\mathbf{T}^{\mathrm{Co}}[\![e_1]\!]H \vee^{Co} \mathbb{P}^{Co}_{\vec{n}}(\overline{c_n})) & \\
 & \quad \Rightarrow \dot{\mathbf{S}}[\![e_2]\!]RO \in \gamma_1^{\mathrm{Co}}(t)\big)\big\} & \\
= & \bigwedge^{Co} \big\{t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \mid \forall (R,O) \in \mathbb{R}' \times \mathbb{O} : & \text{def. } \gamma_1^{\mathrm{Co}}(\cdot) \\
 & \quad \big(\forall \mathtt{x} \in \mathbb{X} \setminus \{\mathtt{f}\} : \mathrm{eval}_{R,O}(\mathtt{x}) \in \gamma_1^{\mathrm{Co}}(H(\mathtt{x})) & \\
 & \quad \wedge\, \mathrm{eval}_{R,O}(\mathtt{f}) \in \gamma_1^{\mathrm{Co}}(H(\mathtt{f}) \wedge^{Co} \big(\mathbf{T}^{\mathrm{Co}}[\![e_1]\!]H \vee^{Co} \mathbb{P}^{Co}_{\vec{n}}(\overline{c_n})\big)) & \\
 & \quad \Rightarrow \dot{\mathbf{S}}[\![e_2]\!]RO \in \gamma_1^{\mathrm{Co}}(t)\big)\big\} & \\
= & \bigwedge^{Co} \big\{t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \mid & \text{def. } \gamma_2^{\mathrm{Co}}(\cdot) \\
 & \quad \forall (R,O) \in \gamma_2^{\mathrm{Co}}(H[\mathtt{f} : H(\mathtt{f}) \wedge^{Co} \big(\mathbf{T}^{\mathrm{Co}}[\![e_1]\!]H \vee^{Co} \mathbb{P}^{Co}_{\vec{n}}(\overline{c_n})\big)]) : & \\
 & \quad \dot{\mathbf{S}}[\![e_2]\!]RO \in \gamma_1^{\mathrm{Co}}(t)\big\} & \\
= & \bigwedge^{Co} \big\{t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \mid & (5.1.2) \\
 & \quad \alpha^{\mathrm{Co}}(\mathbf{C}[\![e_2]\!])H[\mathtt{f} : H(\mathtt{f}) \wedge^{Co} \big(\mathbf{T}^{\mathrm{Co}}[\![e_1]\!]H \vee^{Co} \mathbb{P}^{Co}_{\vec{n}}(\overline{c_n})\big)] \leq^{Co} t\big\} & \\
\leq^{Co} & \bigwedge^{Co} \big\{t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \mid & \text{I.H., trans} \\
 & \quad t = \mathbf{T}^{\mathrm{Co}}[\![e_2]\!]H[\mathtt{f} : H(\mathtt{f}) \wedge^{Co} \big(\mathbf{T}^{\mathrm{Co}}[\![e_1]\!]H \vee^{Co} \mathbb{P}^{Co}_{\vec{n}}(\overline{c_n})\big)]\big\} & \\
\leq^{Co} & \bigwedge^{Co} \big\{\mathbf{T}^{\mathrm{Co}}[\![e_2]\!]H[\mathtt{f} : H(\mathtt{f}) \wedge^{Co} T] \;\big|\; & (5.1.3) \\
 & \quad T = \big(\mathbf{T}^{\mathrm{Co}}[\![e_1]\!]H \vee^{Co} \mathbb{P}^{Co}_{\vec{n}}(\overline{c_n})\big) \wedge & \\
 & \quad H(\mathtt{f}), T \leq^{Co} \{\bot^{\mathrm{Co}} \to \ldots \to \bot^{\mathrm{Co}} \to \emptyset^{\mathrm{Co}}\}\big\} & \\
= & \mathbf{T}^{\mathrm{Co}}[\![\pi \mathtt{f}\ c_1 \ldots c_n = e_1\ \mathtt{in}\ e_2]\!]H & \text{by def.}
\end{array}
$$

The remaining duty of re-establishing monotonicity (3.3.9) is immediate from the definitions of the two new cases. We may conclude:

Theorem 5.1.4. *$\langle \mathbb{T}^{\mathrm{Co}}, \ddot{\leq}^{\mathrm{Co}}, \mathbf{T}^{\mathrm{Co}}[\![\cdot]\!], \gamma^{\mathrm{Co}}, \mathcal{E}^{\mathrm{Co}} \rangle$ is a $\mathbb{Q}$-sound type system, where $\mathcal{E}^{\mathrm{Co}} := \Lambda\theta. \bigcup\{\gamma_2^{\mathrm{Co}}(H) \mid H \in \mathbb{H}^{\mathrm{Co}} \wedge \theta(H) \neq \emptyset^{\mathrm{Co}}\}$.*

5.2 Church/Curry Polytype Semantics for Programs with Overloading

We extend the semantic function $\mathbf{T}^{\mathrm{PC}}[\![\cdot]\!]$ defined in 3.3.3 by definitions for the new syntactic forms:

$$
\begin{array}{lll}
\mathbf{T}^{\mathrm{PC}}[\![\pi \mathtt{f} :: \omega\ \mathtt{in}\ e]\!] & := \big\{\langle H, m\rangle \;\big|\; \langle H[\mathtt{f} : \emptyset], m\rangle \in \mathbf{T}^{\mathrm{PC}}[\![e]\!]\big\} & \\
\mathbf{T}^{\mathrm{PC}}[\![\pi \mathtt{f}\ c_1 \ldots c_n = e_1\ \mathtt{in}\ e_2]\!] & := \big\{\langle H, m\rangle \;\big|\; \exists M \subseteq \mathbb{M}^{PC}_{\vec{n}}(c_1, \ldots, c_n) : \big(& \\
 & \qquad M \neq \emptyset \wedge \forall m' \in M : \langle H, m'\rangle \in \mathbf{T}^{\mathrm{PC}}[\![e_1]\!] & \\
 & \qquad \wedge \langle H[\mathtt{f} : H(\mathtt{f}) \cup M], m\rangle \in \mathbf{T}^{\mathrm{PC}}[\![e_2]\!]\big)\big\} &
\end{array}
$$

5.2.1 Abstraction from Type Collecting Semantics

As our goal is to use the Church/Curry semantics as a step towards the new type system, we immediately consider the two cases we need to check to establish $\mathbb{Q}$-soundness by showing that Lemma 3.3.14 holds for the extended language.

$$
\begin{aligned}
&\alpha^{\mathrm{Po}}(\mathbf{T}^{\mathrm{Co}}[\![\pi \mathtt{f} :: \omega \ \mathtt{in}\ e]\!]) \\
&= \alpha^{\mathrm{Po}}(\Lambda H.\mathbf{T}^{\mathrm{Co}}[\![e]\!]H[\mathtt{f} : \emptyset^{\mathrm{Co}}]) && \text{def. } \mathbf{T}^{\mathrm{Co}}[\![\cdot]\!] \\
&= \left\{\langle H, m\rangle \mid \mathbf{T}^{\mathrm{Co}}[\![e]\!]\gamma_3^{\mathrm{Po}}(H)[\mathtt{f} : \emptyset^{\mathrm{Co}}] \leq^{Co} \gamma_1^{\mathrm{Po}}(m)\right\} && \text{def. } \alpha^{\mathrm{Po}}(\cdot) \\
&= \left\{\langle H, m\rangle \mid \mathbf{T}^{\mathrm{Co}}[\![e]\!](\Lambda \mathtt{x}.\gamma_2^{\mathrm{Po}}(H(\mathtt{x})))[\mathtt{f} : \emptyset^{\mathrm{Co}}] \leq^{Co} \gamma_1^{\mathrm{Po}}(m)\right\} && \text{def. } \gamma_3^{\mathrm{Po}}(\cdot) \\
&= \left\{\langle H, m\rangle \mid \mathbf{T}^{\mathrm{Co}}[\![e]\!](\Lambda \mathtt{x}.\gamma_2^{\mathrm{Po}}(H[\mathtt{f} : \emptyset](\mathtt{x}))) \leq^{Co} \gamma_1^{\mathrm{Po}}(m)\right\} && \text{def. } \gamma_2^{\mathrm{Po}}(\cdot) \\
&= \left\{\langle H, m\rangle \mid \mathbf{T}^{\mathrm{Co}}[\![e]\!]\gamma_3^{\mathrm{Po}}(H[\mathtt{f} : \emptyset]) \leq^{Co} \gamma_1^{\mathrm{Po}}(m)\right\} && \text{def. } \gamma_3^{\mathrm{Po}}(\cdot) \\
&\supseteq \left\{\langle H, m\rangle \mid \gamma^{\mathrm{Po}}(\mathbf{T}^{\mathrm{PC}}[\![e]\!])\gamma_3^{\mathrm{Po}}(H[\mathtt{f} : \emptyset]) \leq^{Co} \gamma_1^{\mathrm{Po}}(m)\right\} && \text{I.H.} \\
&= \Big\{\langle H, m\rangle \mid \\
&\quad \Big(\ddot{\bigwedge}^{Co}\{\gamma_4^{\mathrm{Po}}(\langle H', m'\rangle) \mid \langle H', m'\rangle \in \mathbf{T}^{\mathrm{PC}}[\![e]\!]\}\Big)\gamma_3^{\mathrm{Po}}(H[\mathtt{f} : \emptyset]) \leq^{Co} \gamma_1^{\mathrm{Po}}(m)\Big\} && \text{def. } \ddot{\bigwedge}^{Co}, \gamma_4^{\mathrm{Po}}(\cdot) \\
&= \Big\{\langle H, m\rangle \mid \\
&\quad \Big(\Lambda H'.\bigwedge^{Co}\{\gamma_1^{\mathrm{Co}}(m') \mid \langle H', m'\rangle \in \mathbf{T}^{\mathrm{PC}}[\![e]\!] \wedge H' \dot{\leq}^{Co} \gamma_3^{\mathrm{Po}}(H')\}\Big)\gamma_3^{\mathrm{Po}}(H[\mathtt{f} : \emptyset]) \\
&\quad \leq^{Co} \gamma_1^{\mathrm{Po}}(m)\Big\} && \beta\text{-red.} \\
&= \Big\{\langle H, m\rangle \mid \\
&\quad \bigwedge^{Co}\{\gamma_1^{\mathrm{Co}}(m') \mid \langle H', m'\rangle \in \mathbf{T}^{\mathrm{PC}}[\![e]\!] \wedge \gamma_3^{\mathrm{Po}}(H[\mathtt{f} : \emptyset]) \dot{\leq}^{Co} \gamma_3^{\mathrm{Po}}(H')\} \\
&\quad \leq^{Co} \gamma_1^{\mathrm{Po}}(m)\Big\} \\
&\supseteq \left\{\langle H, m\rangle \mid \bigwedge^{Co}\{\gamma_1^{\mathrm{Co}}(m') \mid \langle H', m'\rangle \in \mathbf{T}^{\mathrm{PC}}[\![e]\!] \wedge H[\mathtt{f} : \emptyset] \dot{\subseteq} H'\} \leq^{Co} \gamma_1^{\mathrm{Po}}(m)\right\} && \text{monotonicity} \\
&\supseteq \left\{\langle H, m\rangle \mid \bigwedge^{Co}\{\gamma_1^{\mathrm{Co}}(m') \mid \langle H', m'\rangle \in \mathbf{T}^{\mathrm{PC}}[\![e]\!] \wedge H[\mathtt{f} : \emptyset] = H'\} \leq^{Co} \gamma_1^{\mathrm{Po}}(m)\right\} \\
&\supseteq \left\{\langle H, m\rangle \mid \langle H[\mathtt{f} : \emptyset], m\rangle \in \mathbf{T}^{\mathrm{PC}}[\![e]\!]\right\}
\end{aligned}
$$

Definition 5.2.1 (Monomorphic Guarded Function Types)**.** For a finite sequence of constructor symbols $s = c_1, \ldots, c_n$, define the set of *s-guarded function types* as

$$\mathbb{M}^{PC}_{\vec{n}}(c_1, \ldots, c_n) := \{c_1(\bar{t_1}) \to \ldots \to c_n(\bar{t_n}) \to t_r \mid \bar{t_1}, \ldots, \bar{t_n}, t_r \in \mathbb{M}^{\mathrm{PC}}\}.$$

The following result can be easily verified in the current context, where the only type constructors across all type systems are those for integer and function types. For other sets of type constructors, an adequate correspondence would have to be established in each case.

Proposition 5.2.2. *For any finite sequence of constructor symbols $c_1, \ldots, c_n$ and any $m \in \mathbb{M}^{\mathrm{PC}}$, if $m \in \mathbb{M}^{PC}_{\vec{n}}(c_1, \ldots, c_n)$, then $\mathbb{P}^{Co}_{\vec{n}}(c_1, \ldots, c_n) \leq^{Co} \gamma_1^{\mathrm{Po}}(m)$.*

We present the proof with a slightly simplified definition of $\mathbf{T}^{\mathrm{Co}}[\![\cdot]\!]$ to ease presentation. The omitted constraint is the one responsible for ensuring that

overloading implementations can be typed as functions of the expected arity, and we use this background assumption as $(*)$ when computing $\wedge^{\mathrm{Co}}$ as set union below.

$$
\begin{aligned}
&\alpha^{\mathrm{Po}}(\mathbf{T}^{\mathrm{Co}}[\![\pi \mathtt{f}\ c_1 \dots c_n = e_1\ \mathtt{in}\ e_2]\!]) \\
&= \alpha^{\mathrm{Po}}(\Lambda H.\mathbf{T}^{\mathrm{Co}}[\![e_2]\!]H[\mathtt{f} : H(\mathtt{f}) \wedge^{Co} (\mathbf{T}^{\mathrm{Co}}[\![e_1]\!]H \vee^{Co} \mathbb{P}^{Co}_{\vec{n}}(c_1, \dots, c_n))]) && \text{def. } \mathbf{T}^{\mathrm{Co}}[\![\cdot]\!] \\
&= \Big\{ \langle H, m\rangle \,\big|\, && \text{def. } \alpha^{\mathrm{Po}}(\cdot) \\
&\quad \mathbf{T}^{\mathrm{Co}}[\![e_2]\!]\gamma_3^{\mathrm{Po}}(H)[\mathtt{f} : \gamma_3^{\mathrm{Po}}(H)(\mathtt{f}) \wedge^{Co} (\mathbf{T}^{\mathrm{Co}}[\![e_1]\!]\gamma_3^{\mathrm{Po}}(H) \vee^{Co} \mathbb{P}^{Co}_{\vec{n}}(\overline{c_n}))] \\
&\qquad \leq^{Co} \gamma_1^{\mathrm{Po}}(m) \Big\} \\
&\supseteq \Big\{ \langle H, m\rangle \,\big|\, && \text{I.H.} \\
&\quad \gamma^{\mathrm{Po}}(\mathbf{T}^{\mathrm{PC}}[\![e_2]\!])\gamma_3^{\mathrm{Po}}(H)[\mathtt{f} : \gamma_3^{\mathrm{Po}}(H)(\mathtt{f}) \wedge^{Co} (\mathbf{T}^{\mathrm{Co}}[\![e_1]\!]\gamma_3^{\mathrm{Po}}(H) \vee^{Co} \mathbb{P}^{Co}_{\vec{n}}(\overline{c_n}))] \\
&\qquad \leq^{Co} \gamma_1^{\mathrm{Po}}(m) \Big\} \\
&= \Big\{ \langle H, m\rangle \,\big|\, && \text{def. } \gamma^{\mathrm{Po}}(\cdot) \\
&\quad H'' = \big(\gamma_3^{\mathrm{Po}}(H)[\mathtt{f} : \gamma_3^{\mathrm{Po}}(H)(\mathtt{f}) \wedge^{Co} (\mathbf{T}^{\mathrm{Co}}[\![e_1]\!]\gamma_3^{\mathrm{Po}}(H) \vee^{Co} \mathbb{P}^{Co}_{\vec{n}}(\overline{c_n}))]\big) \wedge \\
&\quad \Big(\Lambda H'. \bigwedge\nolimits^{Co}\{\gamma_1^{\mathrm{Po}}(m') \,|\, \langle H', m'\rangle \in \mathbf{T}^{\mathrm{PC}}[\![e_2]\!] \wedge H' \dot{\leq}^{Co} \gamma_3^{\mathrm{Po}}(H')\}\Big)(H'') \\
&\qquad \leq^{Co} \gamma_1^{\mathrm{Po}}(m) \Big\} \\
&= \Big\{ \langle H, m\rangle \,\big|\, \textstyle\bigwedge^{Co}\{\gamma_1^{\mathrm{Po}}(m') \mid \langle H', m'\rangle \in \mathbf{T}^{\mathrm{PC}}[\![e_2]\!] && \beta\text{-red.} \\
&\quad \wedge \gamma_3^{\mathrm{Po}}(H)[\mathtt{f} : \gamma_3^{\mathrm{Po}}(H)(\mathtt{f}) \wedge^{Co} (\mathbf{T}^{\mathrm{Co}}[\![e_1]\!]\gamma_3^{\mathrm{Po}}(H) \vee^{Co} \mathbb{P}^{Co}_{\vec{n}}(\overline{c_n}))] \dot{\leq}^{Co} \gamma_3^{\mathrm{Po}}(H')\} \\
&\qquad \leq^{Co} \gamma_1^{\mathrm{Po}}(m) \Big\} \\
&\supseteq \Big\{ \langle H, m\rangle \,\big|\, \exists H', m' : \gamma_1^{\mathrm{Po}}(m') \leq^{Co} \gamma_1^{\mathrm{Po}}(m) \wedge \langle H', m'\rangle \in \mathbf{T}^{\mathrm{PC}}[\![e_2]\!] && \text{def. glb} \\
&\quad \wedge \gamma_3^{\mathrm{Po}}(H)[\mathtt{f} : \gamma_3^{\mathrm{Po}}(H)(\mathtt{f}) \wedge^{Co} (\mathbf{T}^{\mathrm{Co}}[\![e_1]\!]\gamma_3^{\mathrm{Po}}(H) \vee^{Co} \mathbb{P}^{Co}_{\vec{n}}(\overline{c_n}))] \dot{\leq}^{Co} \gamma_3^{\mathrm{Po}}(H') \Big\} \\
&= \Big\{ \langle H, m\rangle \,\big|\, \exists H' : \langle H', m\rangle \in \mathbf{T}^{\mathrm{PC}}[\![e_2]\!] \\
&\quad \wedge \gamma_3^{\mathrm{Po}}(H)[\mathtt{f} : \gamma_3^{\mathrm{Po}}(H)(\mathtt{f}) \wedge^{Co} (\mathbf{T}^{\mathrm{Co}}[\![e_1]\!]\gamma_3^{\mathrm{Po}}(H) \vee^{Co} \mathbb{P}^{Co}_{\vec{n}}(\overline{c_n}))] \dot{\leq}^{Co} \gamma_3^{\mathrm{Po}}(H') \Big\} && (3.3.13)
\end{aligned}
$$

By 5.2.2 and I.H. we have

$$
\begin{aligned}
&\mathbf{T}^{\mathrm{Co}}[\![e_1]\!]\gamma_3^{\mathrm{Po}}(H) \vee^{Co} \mathbb{P}^{Co}_{\vec{n}}(\overline{c_n}) \\
\leq^{Co} &\bigwedge\nolimits^{Co}\{\gamma_1^{\mathrm{Po}}(m) \,|\, \langle H, m\rangle \in \mathbf{T}^{\mathrm{PC}}[\![e_1]\!] \wedge m \in \mathbb{M}^{PC}_{\vec{n}}(\overline{c_n})\},
\end{aligned}
$$

thus

$$
\begin{aligned}
\supseteq \Big\{ &\langle H, m\rangle \,\big|\, \exists H' : \langle H', m\rangle \in \mathbf{T}^{\mathrm{PC}}[\![e_2]\!] \\
&\wedge p = \gamma_3^{\mathrm{Po}}(H)(\mathtt{f}) \wedge^{Co} \bigwedge\nolimits^{Co}\{\gamma_1^{\mathrm{Po}}(m) \,|\, \langle H, m\rangle \in \mathbf{T}^{\mathrm{PC}}[\![e_1]\!] \wedge m \in \mathbb{M}^{PC}_{\vec{n}}(\overline{c_n})\} \\
&\wedge \gamma_3^{\mathrm{Po}}(H)[\mathtt{f} : p] \dot{\leq}^{Co} \gamma_3^{\mathrm{Po}}(H') \Big\}
\end{aligned}
$$

$$
\begin{aligned}
&= \Big\{ \langle H, m\rangle \;\Big|\; \exists H' : \langle H', m\rangle \in \mathbf{T}^{\mathrm{PC}}[\![e_2]\!] && \text{def. } \gamma_2^{\mathrm{Po}}(\cdot)\\
&\quad \wedge p = \gamma_3^{\mathrm{Po}}(H)(\mathtt{f}) \wedge^{Co} \gamma_2^{\mathrm{Po}}(\{m \,|\, \langle H, m\rangle \in \mathbf{T}^{\mathrm{PC}}[\![e_1]\!] \wedge m \in \mathbb{M}_{\vec{n}}^{PC}(\overline{c_n})\})\\
&\quad \wedge \gamma_3^{\mathrm{Po}}(H)[\mathtt{f} : p] \mathrel{\dot{\leq}}^{Co} \gamma_3^{\mathrm{Po}}(H') \Big\}\\
&= \Big\{ \langle H, m\rangle \;\Big|\; \exists H' : \langle H', m\rangle \in \mathbf{T}^{\mathrm{PC}}[\![e_2]\!] && (*), \text{ def. } \gamma_3^{\mathrm{Po}}(\cdot)\\
&\quad \wedge \gamma_3^{\mathrm{Po}}(H[\mathtt{f} : H(\mathtt{f}) \cup \{m \,|\, \langle H, m\rangle \in \mathbf{T}^{\mathrm{PC}}[\![e_1]\!] \wedge m \in \mathbb{M}_{\vec{n}}^{PC}(\overline{c_n})\}]) \mathrel{\dot{\leq}}^{Co} \gamma_3^{\mathrm{Po}}(H') \Big\}\\
&\supseteq \Big\{ \langle H, m\rangle \;\Big|\; \exists H' : \exists M \subseteq \mathbb{M}_{\vec{n}}^{PC}(\overline{c_n}) : \big(M \neq \emptyset \wedge \forall m' \in M : \langle H, m'\rangle \in \mathbf{T}^{\mathrm{PC}}[\![e_1]\!] && \exists\text{-intr.}\\
&\quad \wedge \langle H', m\rangle \in \mathbf{T}^{\mathrm{PC}}[\![e_2]\!] \wedge \gamma_3^{\mathrm{Po}}(H[\mathtt{f} : H(\mathtt{f}) \cup M]) \mathrel{\dot{\leq}}^{Co} \gamma_3^{\mathrm{Po}}(H')\big) \Big\}\\
&\supseteq \Big\{ \langle H, m\rangle \;\Big|\; \exists H' : \exists M \subseteq \mathbb{M}_{\vec{n}}^{PC}(\overline{c_n}) : \big(M \neq \emptyset \wedge \forall m' \in M : \langle H, m'\rangle \in \mathbf{T}^{\mathrm{PC}}[\![e_1]\!] && \text{monotonocity}\\
&\quad \wedge \langle H', m\rangle \in \mathbf{T}^{\mathrm{PC}}[\![e_2]\!] \wedge H[\mathtt{f} : H(\mathtt{f}) \cup M] \mathrel{\dot{\subseteq}} H'\big) \Big\}\\
&\supseteq \Big\{ \langle H, m\rangle \;\Big|\; \exists H' : \exists M \subseteq \mathbb{M}_{\vec{n}}^{PC}(\overline{c_n}) : \big(M \neq \emptyset \wedge \forall m' \in M : \langle H, m'\rangle \in \mathbf{T}^{\mathrm{PC}}[\![e_1]\!]\\
&\quad \wedge \langle H', m\rangle \in \mathbf{T}^{\mathrm{PC}}[\![e_2]\!] \wedge H[\mathtt{f} : H(\mathtt{f}) \cup M] = H'\big) \Big\}\\
&= \Big\{ \langle H, m\rangle \;\Big|\; \exists M \subseteq \mathbb{M}_{\vec{n}}^{PC}(c_1, \ldots, c_n) : \big(M \neq \emptyset \wedge \forall m' \in M : \langle H, m'\rangle \in \mathbf{T}^{\mathrm{PC}}[\![e_1]\!]\\
&\quad \wedge \langle H[\mathtt{f} : H(\mathtt{f}) \cup M], m\rangle \in \mathbf{T}^{\mathrm{PC}}[\![e_2]\!]\big) \Big\}\\
&= \mathbf{T}^{\mathrm{PC}}[\![\pi\mathtt{f}\ c_1 \ldots c_n = e_1\ \mathtt{in}\ e_2]\!]
\end{aligned}
$$

Then we may conclude that $\mathbb{T}^{\mathrm{PC}}$ is a $\mathbb{Q}$-sound abstraction.

5.3 The Type System $\mathbb{T}^{\mathrm{T}}$ as an Abstract Semantics

We come back to the question of soundness of the type system $\mathbb{T}^{\mathrm{T}}$ for the dynamic semantic with overloading. We first give an alternative abstract semantics formulation of the rule-based type system from 4.3.8. Then, by 3.3.6 and 3.3.3, it suffices to show that this abstract semantics abstracts Church/Curry polytype semantics, and we get:

Theorem 5.3.1. *The type system $\mathbb{T}^{\mathrm{T}}$ is a sound type system.*

The re-presentation of the type system as an abstract semantics carries no surprises. Our goal is to give types as sets of typings,

$$\mathbb{T}^{\mathrm{T}} := \mathcal{P}((\mathbb{H}^{\mathrm{T}} \times \mathbb{M}^{\mathrm{T}}))$$

and an abstract semantic function $\mathbf{T}^{\mathrm{T}}[\![\cdot]\!] : \mathbb{E} \to \mathbb{T}^{\mathrm{T}}$, which merely expresses the rule based system in functional form:

$$\mathbf{T}^{\mathrm{T}}[\![\mathtt{x}]\!] := \big\{ \langle \Gamma, \tau\rangle \,\big|\, \tau \leq^{\mathrm{T}} \mathbf{elim}(\Gamma(\mathtt{x})) \wedge \Gamma \in \mathbb{H}^{\mathrm{T}} \big\}$$

$$
\begin{aligned}
&\mathbf{T}^{\mathrm{T}}[\![\lambda \mathtt{x}.e]\!] := \left\{ \langle \Gamma, \tau_1 \to \tau_2 \rangle \;\middle|\; \langle \Gamma[\mathtt{x} : \tau_1], \tau_2 \rangle \in \mathbf{T}^{\mathrm{T}}[\![e]\!] \right\} \\
&\mathbf{T}^{\mathrm{T}}[\![e_1 e_2]\!] := \\
&\qquad \left\{ \langle \Gamma, \tau_2 \rangle \;\middle|\; \langle \Gamma, \tau_1 \to \tau_2 \rangle \in \mathbf{T}^{\mathrm{T}}[\![e_1]\!] \wedge \langle \Gamma, \tau_1 \rangle \in \mathbf{T}^{\mathrm{T}}[\![e_2]\!] \right\} \\
&\mathbf{T}^{\mathrm{T}}[\![\mathtt{let\ x} = e_1 \ \mathtt{in}\ e_2]\!] := \\
&\qquad \left\{ \langle \Gamma, \tau_2 \rangle \;\middle|\; \langle \Gamma, \tau_1 \rangle \in \mathbf{T}^{\mathrm{T}}[\![e_1]\!] \wedge \langle \Gamma[\mathtt{x} : \mathbf{gen}(\Gamma, \tau_1)], \tau_2 \rangle \in \mathbf{T}^{\mathrm{T}}[\![e_2]\!] \right\} \\
&\mathbf{T}^{\mathrm{T}}[\![\mu \mathtt{f}.\lambda \mathtt{x}.e]\!] := \\
&\qquad \left\{ \langle \Gamma, \tau_1 \to \tau_2 \rangle \;\middle|\; \langle \Gamma[\mathtt{f} : \tau_1 \to \tau_2], \tau_1 \to \tau_2 \rangle \in \mathbf{T}^{\mathrm{T}}[\![\lambda \mathtt{x}.e]\!] \right\} \\
&\mathbf{T}^{\mathrm{T}}[\![\pi \mathtt{f} :: \omega \ \mathtt{in}\ e]\!] := \left\{ \langle \Gamma, \tau \rangle \;\middle|\; \langle \Gamma[\mathtt{f} : \ddot{\forall}' \mathtt{a}_\emptyset.[' \mathtt{a}_\emptyset / \$]\omega], \tau \rangle \in \mathbf{T}^{\mathrm{T}}[\![e]\!] \right\} \\
&\mathbf{T}^{\mathrm{T}}[\![\pi \mathtt{f}\ c_1 \ldots c_n = e_1 \ \mathtt{in}\ e_2]\!] := \\
&\qquad \Big\{ \langle \Gamma[\mathtt{f} : \ddot{\forall}' \mathtt{a}_X.\tau_0], \tau_2 \rangle \;\Big|\; \exists \tau_1 : \exists c \in \mathsf{C} : \big(\\
&\qquad\quad \langle \Gamma[\mathtt{f} : \ddot{\forall}' \mathtt{a}_X.\tau_0], \tau_1 \rangle \in \mathbf{T}^{\mathrm{T}}[\![e_1]\!] \\
&\qquad\quad \wedge\ \tau_1 \leq^{\mathrm{T}} c_1(\overline{' \mathtt{a}^{\mathtt{1}}_{\top}}) \to \ldots \to c_n(\overline{' \mathtt{a}^{\mathtt{n}}_{\top}}) \to ' \mathtt{b}_{\top} \\
&\qquad\quad \wedge\ \tau_1 = [c(\overline{' \mathtt{c}})/' \mathtt{a}_X]\tau_0 \\
&\qquad\quad \wedge\ c \notin d \\
&\qquad\quad \wedge\ \langle \Gamma[\mathtt{f} : \ddot{\forall}' \mathtt{a}_{X \cup \{c(\overline{' \mathtt{c}})\}}.[' \mathtt{a}_{X \cup \{c(\overline{' \mathtt{c}})\}} / ' \mathtt{a}_X]\tau_0], \tau_2 \rangle \in \mathbf{T}^{\mathrm{T}}[\![e_2]\!] \big) \Big\} \\
&\mathbf{T}^{\mathrm{T}}[\![z]\!] := \left\{ \langle \Gamma, \mathtt{int} \rangle \;\middle|\; \Gamma \in \mathbb{H}^{\mathrm{T}} \right\} \\
&\mathbf{T}^{\mathrm{T}}[\![e_1 + e_2]\!] := \left\{ \langle \Gamma, \mathtt{int} \rangle \;\middle|\; \langle \Gamma, \mathtt{int} \rangle \in \mathbf{T}^{\mathrm{T}}[\![e_1]\!] \cap \mathbf{T}^{\mathrm{T}}[\![e_2]\!] \right\} \\
&\mathbf{T}^{\mathrm{T}}[\![\mathtt{ifz}\ e_1 \ \mathtt{then}\ e_2 \ \mathtt{else}\ e_3]\!] := \\
&\qquad \left\{ \langle \Gamma, \tau \rangle \;\middle|\; \langle \Gamma, \mathtt{int} \rangle \in \mathbf{T}^{\mathrm{T}}[\![e_1]\!] \wedge \langle \Gamma, \tau \rangle \in \mathbf{T}^{\mathrm{T}}[\![e_2]\!] \cap \mathbf{T}^{\mathrm{T}}[\![e_3]\!] \right\}
\end{aligned}
$$

Definition 5.3.2. For converting from type schemes to types we define

$$
\begin{aligned}
&\mathbf{elim} : \mathbb{P}^{\mathrm{T}} \to \mathbb{M}^{\mathrm{T}} \\
&\mathbf{elim}(\tau) := \tau \\
&\mathbf{elim}(\dot{\forall}' \mathtt{b}_1 \ldots ' \mathtt{b}_\mathtt{n}.\tau) := \tau \\
&\mathbf{elim}(\ddot{\forall}' \mathtt{b}_1 \ldots ' \mathtt{b}_\mathtt{n}.\tau) := \tau.
\end{aligned}
$$

We need operations on type terms similar to those discussed in [Cou97, p. 321], but adjusted to the notion of respectful substitution, such that the set of groundings of a given term contains only those obtained by applying *respectful* substitutions.

Definition 5.3.3. $\mathbf{ground} : \mathbb{M}^{\mathrm{T}} \to \mathbb{P}^{\mathrm{PC}}$ takes type terms with variables to sets of ground types.

$$
\mathbf{ground}(\tau) := \begin{cases} \emptyset & \text{if } \exists ' \mathtt{a}_\emptyset \in \mathbf{FV}(\tau) \\ \{S(\tau) \mid S : \mathbb{M}^{\mathrm{T}} \xrightarrow{r} \mathbb{M}^{\mathrm{PC}}\} & \text{o.w.} \end{cases}
$$

The test for a $' \mathtt{a}_\emptyset$ appearing freely in τ is necessary to make **ground** be total, as any such variable is uninstantiable.

For the purpose of proving this semantics $\mathbb{Q}$-sound, we first define a concretization function for type environments $\dot{\gamma}^{\mathrm{T}} : \mathbb{H}^{\mathrm{T}} \to \mathcal{P}(\mathbb{H}^{\mathrm{PC}})$, which unfolds all polytypes in $\mathbb{H}^{\mathrm{T}}$ type environments to their (potentially) infinite set representations after grounding variables bound globally in the environment first:

$$\dot{\gamma}^{\mathrm{T}}(\Gamma) := \left\{ \Lambda \mathtt{x}.\mathbf{ground}(\mathbf{elim}(S(\Gamma)(\mathtt{x}))) \,\middle|\, S : \mathbb{V} \xrightarrow{r} \mathbb{M}^{\mathrm{PC}} \right\}$$

Observe that $\dot{\gamma}^{\mathrm{T}}$ produces singleton sets for type environments without parametric variables. We therefore permit ourselves to use $\dot{\gamma}^{\mathrm{T}}$ as if it simply produced a type environment in this case. We can extend the order on type terms to type environments in pointwise fashion:

$$\Gamma \dot{\leq}^{\mathrm{T}} \Gamma' :\Leftrightarrow \forall \mathtt{x} \in \mathbb{X} : \forall S : \mathbb{V} \xrightarrow{r} \mathbb{M}^{\mathrm{PC}} : \Gamma(\mathtt{x}) \leq^{\mathrm{T}} \Gamma'(\mathtt{x})$$

Definition 5.3.4. We define a function $\ddot{\gamma}^{\mathrm{T}} : \mathbb{T}^{\mathrm{T}} \to \mathbb{T}^{\mathrm{PC}}$ unfolding finitely represented types to their infinite expansion,

$$\ddot{\gamma}^{\mathrm{T}}(T) := \left\{ \langle \dot{\gamma}^{\mathrm{T}}(S(\Gamma)), S(\tau) \rangle \,\middle|\, \langle S(\Gamma), S(\tau) \rangle \in T \wedge S : \mathbb{V} \xrightarrow{r} \mathbb{M}^{\mathrm{PC}} \right\}.$$

Note that in terms of τ this generates all ground instances, but we simultaneously ground free variables in the type environment so that the coherence of global type variables is ensured.

5.3.1 Abstraction from Church/Curry Polytype Semantics

Proposition 5.3.5. *$\langle \mathbb{T}^{\mathrm{T}}, \subseteq \rangle$ is an abstraction of $\langle \mathbb{T}^{\mathrm{PC}}, \subseteq \rangle$ via $\ddot{\gamma}^{\mathrm{T}}$.*

Proof. The monotonicity of $\ddot{\gamma}^{\mathrm{T}}$ is straightforward. By (3.3.2) it remains to check that for all $e \in \mathbb{E}$

$$\mathbf{T}^{\mathrm{PC}}[\![e]\!] \supseteq \ddot{\gamma}^{\mathrm{T}}(\mathbf{T}^{\mathrm{T}}[\![e]\!]).$$

The proof is by induction on the structure of e. We first consider the variable case.

$$\begin{aligned}
&\ddot{\gamma}^{\mathrm{T}}(\mathbf{T}^{\mathrm{T}}[\![\mathtt{x}]\!]) \\
&= \ddot{\gamma}^{\mathrm{T}}\left(\left\{ \langle \Gamma, \tau \rangle \,\middle|\, \tau \leq^{\mathrm{T}} \mathbf{elim}(\Gamma(\mathtt{x})) \wedge \Gamma \in \mathbb{H}^{\mathrm{T}} \right\}\right) && \text{def. } \mathbf{T}^{\mathrm{T}}[\![\cdot]\!] \\
&= \left\{ \langle \dot{\gamma}^{\mathrm{T}}(S(\Gamma)), S(\tau) \rangle \,\middle|\, S(\tau) \leq^{\mathrm{T}} \mathbf{elim}(S(\Gamma(\mathtt{x}))) \wedge S(\Gamma) \in \mathbb{H}^{\mathrm{T}} \right. && \text{def. } \ddot{\gamma}^{\mathrm{T}} \\
&\qquad \left. \wedge\, S \in \mathbb{V} \xrightarrow{r} \mathbb{M}^{\mathrm{PC}} \right\} \\
&= \left\{ \langle \dot{\gamma}^{\mathrm{T}}(S(\Gamma)), S(\tau) \rangle \,\middle|\, S(\tau) \in \dot{\gamma}^{\mathrm{T}}(S(\Gamma))(\mathtt{x}) \wedge S(\Gamma) \in \mathbb{H}^{\mathrm{T}} \right. && \text{def. } \dot{\gamma}^{\mathrm{T}} \\
&\qquad \left. \wedge\, S \in \mathbb{V} \xrightarrow{r} \mathbb{M}^{\mathrm{PC}} \right\} \\
&\subseteq \left\{ \langle H, m \rangle \,\middle|\, m \in H(\mathtt{x}) \wedge H \in \mathbb{H}^{\mathrm{PC}} \right\} \\
&= \mathbf{T}^{\mathrm{PC}}[\![\mathtt{x}]\!] && \text{def. } \mathbf{T}^{\mathrm{PC}}[\![\cdot]\!]
\end{aligned}$$

The other interesting cases are those for overloading introduction and implementation. The introduction of overloaded identifiers is relatively straightforward

and relies on the fact that groundings of type terms containing a type variable $'\mathtt{a}_\emptyset$ produce the empty set.

$$
\begin{aligned}
&\ddot{\gamma}^{\mathrm{T}}(\mathbf{T}^{\mathrm{T}}[\![\pi \mathtt{f} :: \omega \ \mathtt{in}\ e]\!]) \\
&= \ddot{\gamma}^{\mathrm{T}}\big(\big\{\langle \Gamma, \tau\rangle \,\big|\, \langle \Gamma[\mathtt{f} : \ddot{\forall}'\mathtt{a}_\emptyset.['\mathtt{a}_\emptyset/\$]\omega], \tau\rangle \in \mathbf{T}^{\mathrm{T}}[\![e]\!]\big\}\big) \\
&= \big\{\langle \dot{\gamma}^{\mathrm{T}}(S(\Gamma)), S(\tau)\rangle \,\big|\, \\
&\qquad \langle S(\Gamma[\mathtt{f} : \ddot{\forall}'\mathtt{a}_\emptyset.['\mathtt{a}_\emptyset/\$]\omega]), S(\tau)\rangle \in \mathbf{T}^{\mathrm{T}}[\![e]\!] \wedge S : \mathbb{V} \xrightarrow{r} \mathbb{M}^{\mathrm{PC}}\big\} \\
&\subseteq \big\{\langle \dot{\gamma}^{\mathrm{T}}(S(\Gamma)), S(\tau)\rangle \,\big|\, \\
&\qquad \langle \dot{\gamma}^{\mathrm{T}}(S(\Gamma[\mathtt{f} : \ddot{\forall}'\mathtt{a}_\emptyset.['\mathtt{a}_\emptyset/\$]\omega])), S(\tau)\rangle \in \mathbf{T}^{\mathrm{PC}}[\![e]\!] \wedge S : \mathbb{V} \xrightarrow{r} \mathbb{M}^{\mathrm{PC}}\big\} \\
&= \big\{\langle \dot{\gamma}^{\mathrm{T}}(S(\Gamma)), S(\tau)\rangle \,\big|\, \\
&\qquad \langle \dot{\gamma}^{\mathrm{T}}(S(\Gamma))[\mathtt{f} : \mathbf{ground}(\mathbf{elim}(S(\ddot{\forall}'\mathtt{a}_\emptyset.['\mathtt{a}_\emptyset/\$]\omega)))], S(\tau)\rangle \in \mathbf{T}^{\mathrm{PC}}[\![e]\!] \\
&\qquad \wedge S : \mathbb{V} \xrightarrow{r} \mathbb{M}^{\mathrm{PC}}\big\} \\
&= \big\{\langle \dot{\gamma}^{\mathrm{T}}(S(\Gamma)), S(\tau)\rangle \,\big|\, \langle \dot{\gamma}^{\mathrm{T}}(S(\Gamma))[\mathtt{f} : \emptyset], S(\tau)\rangle \in \mathbf{T}^{\mathrm{PC}}[\![e]\!] \wedge S : \mathbb{V} \xrightarrow{r} \mathbb{M}^{\mathrm{PC}}\big\} \\
&\subseteq \big\{\langle H, m\rangle \,\big|\, \langle H[\mathtt{f} : \emptyset], m\rangle \in \mathbf{T}^{\mathrm{PC}}[\![e]\!]\big\} \\
&= \mathbf{T}^{\mathrm{PC}}[\![\pi \mathtt{f} :: \omega \ \mathtt{in}\ e]\!]
\end{aligned}
$$

For the case of overloading definitions, we need the following proposition which follows by induction on the structure of program expressions.

Proposition 5.3.6. *For any* $e, \Gamma, \tau, S : \mathbb{V} \xrightarrow{r} \mathbb{M}^{\mathrm{PC}}$,

$$\langle S(\Gamma), \tau\rangle \in \mathbf{T}^{\mathrm{T}}[\![e]\!] \Rightarrow \forall S' : \mathbb{V} \xrightarrow{r} \mathbb{M}^{\mathrm{PC}} : \langle S(\Gamma), S'(\tau)\rangle \in \mathbf{T}^{\mathrm{T}}[\![e]\!].$$

$$
\begin{aligned}
&\ddot{\gamma}^{\mathrm{T}}\big(\mathbf{T}^{\mathrm{T}}[\![\pi \mathtt{f}\ c_1 \ldots c_n = e_1 \ \mathtt{in}\ e_2]\!]\big) \\
&= \ddot{\gamma}^{\mathrm{T}}\big(\big\{\langle \Gamma[\mathtt{f} : \ddot{\forall}'\mathtt{a}_X.\tau_0], \tau_2\rangle \,\big|\, \exists \tau_1 : \exists c \in \mathsf{C} : \big(\\
&\quad \langle \Gamma[\mathtt{f} : \ddot{\forall}'\mathtt{a}_X.\tau_0], \tau_1\rangle \in \mathbf{T}^{\mathrm{T}}[\![e_1]\!] \\
&\quad \wedge \tau_1 \leq^{\mathrm{T}} c_1(\overline{'\mathtt{a}^{\mathtt{1}}_{\top}}) \to \ldots \to c_n(\overline{'\mathtt{a}^{\mathtt{n}}_{\top}}) \to '\mathtt{b}_{\top} \\
&\quad \wedge \tau_1 = [c(\overline{'\mathtt{c}})/'\mathtt{a}_X]\tau_0 \\
&\quad \wedge c \notin d \\
&\quad \wedge \langle \Gamma[\mathtt{f} : \ddot{\forall}'\mathtt{a}_{X\cup\{c(\overline{'\mathtt{c}})\}}.['\mathtt{a}_{X\cup\{c(\overline{'\mathtt{c}})\}}/'\mathtt{a}_X]\tau_0], \tau_2\rangle \in \mathbf{T}^{\mathrm{T}}[\![e_2]\!]\big)\big\}\big) \\
&= \big\{\langle \dot{\gamma}^{\mathrm{T}}(S(\Gamma[\mathtt{f} : \ddot{\forall}'\mathtt{a}_X.\tau_0])), S(\tau_2)\rangle \,\big|\, \exists \tau_1 : \exists c \in \mathsf{C} : \big(\\
&\quad \langle S(\Gamma[\mathtt{f} : \ddot{\forall}'\mathtt{a}_X.\tau_0]), \tau_1\rangle \in \mathbf{T}^{\mathrm{T}}[\![e_1]\!] \\
&\quad \wedge \tau_1 \leq^{\mathrm{T}} c_1(\overline{'\mathtt{a}^{\mathtt{1}}_{\top}}) \to \ldots \to c_n(\overline{'\mathtt{a}^{\mathtt{n}}_{\top}}) \to '\mathtt{b}_{\top} \\
&\quad \wedge \tau_1 = [c(\overline{'\mathtt{c}})/'\mathtt{a}_X]\tau_0 \\
&\quad \wedge c \notin d \\
&\quad \wedge \langle S(\Gamma[\mathtt{f} : \ddot{\forall}'\mathtt{a}_{X\cup\{c(\overline{'\mathtt{c}})\}}.['\mathtt{a}_{X\cup\{c(\overline{'\mathtt{c}})\}}/'\mathtt{a}_X]\tau_0]), S(\tau_2)\rangle \in \mathbf{T}^{\mathrm{T}}[\![e_2]\!]\big) \\
&\quad \wedge S : \mathbb{V} \xrightarrow{r} \mathbb{M}^{\mathrm{PC}}\big\} \\
&\subseteq \big\{\langle \dot{\gamma}^{\mathrm{T}}(S(\Gamma[\mathtt{f} : \ddot{\forall}'\mathtt{a}_X.\tau_0])), S(\tau_2)\rangle \,\big|\, \exists \tau_1 : \exists c \in \mathsf{C} : \big(
\end{aligned}
$$

$$
\begin{aligned}
&\forall m \in \mathbf{ground}(\tau_1) : \langle \dot\gamma^{\mathrm{T}} S(\Gamma[\mathtt{f} : \ddot\forall' \mathtt{a}_X.\tau_0]), m\rangle \in \mathbf{T}^{\mathrm{PC}}[\![e_1]\!] \\
&\wedge \mathbf{ground}(\tau_1) \subseteq \mathbb{M}^{PC}_{\vec n}(c_1,\ldots,c_n) \\
&\wedge \tau_1 = [c(\overline{'\mathtt{c}})/'\mathtt{a}_X]\tau_0 \\
&\wedge c \notin d \\
&\wedge \langle S(\Gamma[\mathtt{f} : \ddot\forall' \mathtt{a}_{X\cup\{c(\overline{'\mathtt{c}})\}}.['\mathtt{a}_{X\cup\{c(\overline{'\mathtt{c}})\}}/'\mathtt{a}_X]\tau_0]), S(\tau_2)\rangle \in \mathbf{T}^{\mathrm{T}}[\![e_2]\!]\big) \\
&\wedge S : \mathbb{V} \xrightarrow{r} \mathbb{M}^{\mathrm{PC}}\big\} \\
\subseteq \big\{\langle \dot\gamma^{\mathrm{T}}(S(\Gamma[\mathtt{f} : \ddot\forall' \mathtt{a}_X.\tau_0])), S(\tau_2)\rangle \,\big|\, \exists \tau_1 : \exists c \in \mathsf{C} : \big(\\
&\forall m \in \mathbf{ground}(\tau_1) : \langle \dot\gamma^{\mathrm{T}} S(\Gamma[\mathtt{f} : \ddot\forall' \mathtt{a}_X.\tau_0]), m\rangle \in \mathbf{T}^{\mathrm{PC}}[\![e_1]\!] \\
&\wedge \mathbf{ground}(\tau_1) \subseteq \mathbb{M}^{PC}_{\vec n}(c_1,\ldots,c_n) \\
&\wedge \tau_1 = [c(\overline{'\mathtt{c}})/'\mathtt{a}_X]\tau_0 \\
&\wedge c \notin d \\
&\wedge \langle \dot\gamma^{\mathrm{T}}(S(\Gamma[\mathtt{f} : \ddot\forall' \mathtt{a}_{X\cup\{c(\overline{'\mathtt{c}})\}}.['\mathtt{a}_{X\cup\{c(\overline{'\mathtt{c}})\}}/'\mathtt{a}_X]\tau_0])), S(\tau_2)\rangle \in \mathbf{T}^{\mathrm{PC}}[\![e_2]\!]\big) \\
&\wedge S : \mathbb{V} \xrightarrow{r} \mathbb{M}^{\mathrm{PC}}\big\}
\end{aligned}
$$

But now we have that

$$
\begin{aligned}
&\dot\gamma^{\mathrm{T}}(S(\Gamma[\mathtt{f} : \ddot\forall' \mathtt{a}_{X\cup\{c(\overline{'\mathtt{c}})\}}.['\mathtt{a}_{X\cup\{c(\overline{'\mathtt{c}})\}}/'\mathtt{a}_X]\tau_0])) \\
=&\dot\gamma^{\mathrm{T}}(S(\Gamma))[\mathtt{f} : \mathbf{ground}(\mathbf{elim}(S(\ddot\forall' \mathtt{a}_{X\cup\{c(\overline{'\mathtt{c}})\}}.['\mathtt{a}_{X\cup\{c(\overline{'\mathtt{c}})\}}/'\mathtt{a}_X]\tau_0)))] \\
=&\dot\gamma^{\mathrm{T}}(S(\Gamma))[\mathtt{f} : \mathbf{ground}(\mathbf{elim}(S(\ddot\forall' \mathtt{a}_X.\tau_0))) \cup \mathbf{ground}([c(\overline{'\mathtt{c}})/'\mathtt{a}_X]\tau_0)] \\
=&\dot\gamma^{\mathrm{T}}(S(\Gamma))[\mathtt{f} : \mathbf{ground}(\mathbf{elim}(S(\ddot\forall' \mathtt{a}_X.\tau_0))) \cup \mathbf{ground}(\tau_1)] \\
=&\dot\gamma^{\mathrm{T}}(S(\Gamma[\mathtt{f} : \ddot\forall' \mathtt{a}_X.\tau_0]))[\mathtt{f} : \dot\gamma^{\mathrm{T}}(S(\Gamma[\mathtt{f} : \ddot\forall' \mathtt{a}_X.\tau_0]))(\mathtt{f}) \cup \mathbf{ground}(\tau_1)],
\end{aligned}
$$

and so we get an inclusion

$$
\begin{aligned}
&\subseteq \big\{\langle \dot\gamma^{\mathrm{T}}(S(\Gamma[\mathtt{f} : \ddot\forall' \mathtt{a}_X.\tau_0])), S(\tau_2)\rangle \,\big|\, \exists M : \big(\\
&\quad \forall m \in M : \langle \dot\gamma^{\mathrm{T}} S(\Gamma[\mathtt{f} : \ddot\forall' \mathtt{a}_X.\tau_0]), m\rangle \in \mathbf{T}^{\mathrm{PC}}[\![e_1]\!] \\
&\quad \wedge M \subseteq \mathbb{M}^{PC}_{\vec n}(c_1,\ldots,c_n) \wedge M \neq \emptyset \\
&\quad \wedge \langle \dot\gamma^{\mathrm{T}}(S(\Gamma[\mathtt{f} : \ddot\forall' \mathtt{a}_X.\tau_0]))[\mathtt{f} : \dot\gamma^{\mathrm{T}}(S(\Gamma[\mathtt{f} : \ddot\forall' \mathtt{a}_X.\tau_0]))(\mathtt{f}) \cup M], S(\tau_2)\rangle \in \mathbf{T}^{\mathrm{PC}}[\![e_2]\!]\big) \\
&\quad \wedge S : \mathbb{V} \xrightarrow{r} \mathbb{M}^{\mathrm{PC}}\big\} \\
&\subseteq \mathbf{T}^{\mathrm{PC}}[\![\pi\mathtt{f}\ c_1 \ldots c_n = e_1\ \mathtt{in}\ e_2]\!].
\end{aligned}
$$

□

Chapter 6

Related Work

> Ernest was an elephant and very well intentioned,
> Leonard was a lion with a brave new tail,
> George was a goat, as I think I have mentioned,
> but James was only a snail.
>
> —A.A. Milne, *The Four Friends*

Our work is an attempt at establishing a connection between a variety of topics in programming languages, where the main strands are dynamic dispatch (or, definition by cases), static analysis via abstract interpretation, and bounded polymorphism. In our experience, connections between these areas are manifold and enlightening, yet the differences in outlook and weak connections between their respective research communities present a challenge to their combined study. We will go into each topic individually.

6.1 Bounded Polymorphism

The most obvious and direct connection is with the work of Stefan Kaes, whose work on *parametric overloading* [Kae88] was one of the earliest systematic approaches to bounded polymorphism. As had been clear in the 1980s, Milner's judgment of overloading being "somewhat orthogonal" to parametric polymorphism was too optimistic[1]. The standard examples of the problems arising from the interaction between overloading and parametricity are the arithmetic and equality operations in *StandardML* [MTH90]. As an example, we may consider the equality case, which is a binary predicate, or binary function `==` mapping two values to a boolean value. Because equality of function values is undecidable, `==` may not be applied to any functions. At the same time, `==` possesses infinitely many instances of the scheme `list(a)` → `list(a)` → `bool`, but not for arbitrary instantiations of `a`, as the implementation of `==` for lists must again rely on `==` to check equality of list members. That is, the `list(a)` → `list(a)` → `bool` family

[1] Maybe depending on what one understands "somewhat" to mean.

of types is parametric, but with a bound on `a` that requires instantiations to again support equality. The ad-hoc solution to this problem in *StandardML* is the introduction of a second sort of variables which may only be unified with an inductively defined set of types guaranteed to admit equality.

The work of Kaes consists in a systematization and generalization of the sorted variable approach. While our system $\mathbb{T}^{\mathrm{T}}$ relies on variables sorted by type annotations, variables in [Kae88, Kae05] are sorted by *operator symbol* annotations. The benefit of this approach is enhanced extensibility of type definitions which allows type assignment to be independent of the order in which overloading implementations are declared. This is achieved by using the operator symbol annotations as an indirection to type annotations via a global look-up table. Kaes's 2005 dissertation [Kae05] elaborates on the results of [Kae88] and generalizes the approach to a theory of constrained types, a development that is reminiscent of the work of Sulzmann et al. which we will discuss below. Kaes's work seems to have had little direct impact, as an independent and parallel development of a similar approach rose to popularity. We believe that especially his 2005 dissertation still offers a worthwhile perspective that has been unduly neglected.

Wadler and Blott's *How to make* ad-hoc *polymorphism less* ad hoc [WB89], which first popularized bounded parametric polymorphism, is another excellent systematization of the ad hoc type system extensions found in *StandardML*. Because their development grew out of and was incorporated into the *Haskell* programming language, it was arguably more accessible and found swift adoption and recognition. The *type class* approach fundamentally follows the same strategy of bounding type variables by sets of constraints, but places less constraints on the shape of overloaded signatures and furthermore allows to *bundle* operator constraints into named predicates (e.g. `Eq`, `Num`), which much improves intelligibility.

Type classes became a popular research topic throughout the 1990s and found extensive elaboration in the work of Mark Jones [Jon92, Jon93, Jon95], whose work was of both theoretical and practical nature through his *Haskell* dialect *Gofer*, and forms the basis for much of what is today commonly understood to fall under the type class approach, most prominently the extension to type *constructors* and multi-parameter type classes. Partial summaries of the history of type classes in *Haskell* can be found in [JJM97] and [HHJW07].

As the type class approach proved to be a success in both theory and practice, multiple further generalizations of the combination of Hindley-Milner type inference with type constraint have been proposed. The longest standing development begins with *Type Inference with Constrained Types* [OSW97] and has found its way back into the *Haskell* compiler *GHC* as described in [VJSS11]. The basic idea is to parametrize Hindley-Milner type inference by a constraint system X to obtain a general procedure HM(X) which under specific conditions on the constraint system may rely on a uniform type inference algorithm and guarantee principle types. The current iteration OutsideIn(X) is both very powerful and complicated, as is attested to by the 78-page spanning [VJSS11]. An introduction to the HM(X) approach can be found in [PR05].

6.2 Dynamic Dispatch

While our dynamic language λ_{K} and its semantics are inspired by one of the compilation strategies suggest by Kaes, it is arguably more interesting to attempt an adaptation of the type class approach to dynamic languages. This is done in *Type Classes Without Types* [GL05], which defines the notion of *predicate classes* and presents an implementation for the *Scheme* programming language. For lack of a static type system, the type class model is adapted to make use of the *latent types* associated with values at run time by means of predicate functions, which is a form of dynamic dispatch, but generalized to dispatch on arbitrary predicates instead of just tags on values. Essentially the mechanism allows to define overloadings by cases, but as in our semantics, a global look-up table is employed in order to maintain openness and extensibility of overloading implementations. An added benefit inherited from type classes is the ability to group operators semantically. It would be interesting to consider the problem of type inference for this or a similar dynamic adaptation of the type class approach.

Although the type system $\mathbb{T}^{\mathrm{T}}$ forces dispatch on only one type, our *dynamic* semantics affords *multiple dispatch*. As the name suggest, *multiple dispatch* makes use of an analysis of several arguments in order to determine the implementation to dispatch to. Besides efficiency challenges in implementing multiple dispatch, additional challenges are posed for static type inference. We have not considered the problem of reconciling multiple dispatch and static analysis. [MPTN08] is an empirical study of practical use of multiple dispatch across several languages and describes some interactions between multiple dispatch and static type systems.

6.3 Abstract Interpretation and Type Analysis

We briefly discuss two examples of extensions of [Cou97]. Roberta Gori and Giorgio Levi use the abstract interpretation framework for a purpose suggested by Cousot: "A sound type inference algorithm which would be more precise than required by the typing rules would be harmless for the programmer and certainly useful to an optimizing compiler" [Cou97, p. 330]. By analyzing the Damas-Milner [DM82] approach to typing recursive definitions using the lattice theoretic viewpoint of abstract interpretation, Gori and Levi identify it as an application of a *widening operation* used to efficiently approximate an upper approximation of the least fixed point. Using this insight, a generalization of this approach is derived which may be used to increase the precision of the approximation and generates a type system which in precision lies between systems for monomorphic and polymorphic recursion [GL02, GL03].

More recently, Simon uses the abstract interpretation framework to combine type inference with inference for the size of vectors [Sim14]. Simon's development is motivated by the desire to derive type systems for domain-specific languages without starting from scratch. The use of static analysis methods

from abstract interpretation promises to allow for reuse of existing abstract domains, as well as alternative representations of types instead of just Herbrand-style type terms. Simon discusses difficulties with Cousot's presentation of the Milner-Mycroft type system for polymorphic recursion, which however was not the focus of our research.

Chapter 7

Conclusion

> We must face the fact that we are on the brink of times when man may be able to magnify his intellectual and inventive capability, just as in the nineteenth century he used machines to magnify his physical capacity. Again, as then, our innocence is lost. And again, of course, the innocence, once lost, cannot be regained. The loss demands attention, not denial.
>
> —Christopher Alexander, *Notes on the Synthesis of Form*

To conclude, we want to summarize our contributions, discuss benefits of an abstract interpretation approach as well as possibilities for future work, and finally connect back to the motivating goal of reconciliating type theory and engineering practice.

We defined an applicative language with dynamic dispatch and its denotational semantics, which though based on prior work by Kaes ([Kae88, Kae05]) is novel in that it demonstrates the optional nature of type inference for the purpose of compilation. While we have not discussed this in detail, this language has been designed to be compatible with Kaes's original type inference procedure and we believe that indeed Kaes's type system could be applied to this language without major modifications. We furthermore designed a novel type system for said language which introduces a system of local constraints on type variables to support a simple form of bounded type polymorphism. We have devised a unification procedure for this system which may be substituted for Robinson unification to obtain a type inference procedure along the lines of Damas-Milner type inference. Finally, we have proved soundness of two type systems adapted from [Cou97] and used this development to prove soundness of the novel type system by abstraction.

We claim that the abstract interpretation perspective has the following benefits:

- It may help to broaden the understanding of what it means to create a type system using term-based representations of infinite sets of types by providing a framework in which both practical type systems and their

intended idealizations may be formalized and brought into well-defined relations.

- It provides a set of tools for deriving sound, implementable abstractions from idealized systems.
- It provides a common framework for both type-based analysis and other static analysis methods, such that there is potential for combining methods from both.

Many possibilities for future work come to mind. In direct extension of our work, the use of external type systems for the purpose of optimizing compilation could be discussed—for example, knowledge of the precise type of a function at call-site could be used to bypass dynamic dispatch by directly rewriting the overloaded application to the corresponding implementation.

We were unfortunately unable to finish a full abstract interpretation treatment of Kaes's original type system in time for the thesis deadline, and would see this, or other systems for bounded polymorphism, as an interesting object of study that has not been considered from an abstract interpretation viewpoint. Similarly, the connection between gradual typing and abstract-interpretation-based analysis methods has to our knowledge not been made.

Lastly, the connection between bounded polymorphism and order-sorted unification [MGS89] has been noticed by Kaes [Kae05] and others, but the abstract interpretation perspective suggests to consider the opposite transformation from concrete to abstract semantics via order-sorted anti-unification [AEMO09].

In closing, since becoming aware of the mismatch between programming language theory rhetoric and industry practice during the early stages of this thesis, we have noticed many encouraging signals coming from both academia and industry that give cause for an optimistic outlook on a future convergence of programming language theory and practice.

Appendix A

Some Errors in [Cou97]

A.1 The Soundness of $\mathbf{T}^{\mathrm{Co}}[\![\cdot]\!]$

$\mathbf{T}^{\mathrm{Co}}[\![\cdot]\!]$ as defined in [Cou97, p. 324] is unsound and Proposition 15 (p. 325) can not be shown to hold. We give counterexamples to Proposition 15 and suggest a corrected definition of $\mathbf{T}^{\mathrm{Co}}[\![\cdot]\!]$ for which the proposition holds.

Proposition A.1.1. *For all* $t \in \mathbb{P}^{\mathrm{Co}}_{\equiv}$, $\bot \in \gamma_1^{\mathrm{Co}}(t)$.

Proposition A.1.2. *For all* $t \in \mathbb{P}^{\mathrm{Co}}_{\equiv}$, $\boldsymbol{\times} \in \gamma_1^{\mathrm{Co}}(t)$ *if and only if* $t = \emptyset^{\mathrm{Co}}$.

Proposition A.1.3. *For all* $T, S \subseteq \mathbb{P}^{\mathrm{Co}}_{\equiv}$, $T \subseteq S$ *implies* $\bigwedge^{Co} S \leq^{Co} \bigwedge^{Co} T$.

Proposition A.1.4. *For all* $t \in \mathbb{P}^{\mathrm{Co}}_{\equiv}, u \in \mathbb{U}, z \in \mathbb{Z}_\bot$, *if* $u \notin \mathbb{Z}_\bot$ *and* $u, z \in \gamma_1^{\mathrm{Co}}(t)$, *then* $t = \emptyset^{\mathrm{Co}}$.

Inconsistency 1: $\mathbf{T}^{\mathrm{Co}}[\![\lambda \mathtt{x}.e]\!]$ The rule for typing lambda abstractions introduces an inconsistency as it allows for expressions with runtime errors to be typed. We hypothesize that this is merely an unfortunate typographical error. As an example, consider the term $\phi = (\lambda \mathtt{x.1})(\mathtt{if}\ (\lambda \mathtt{x.x})\ \mathtt{then\ 1\ else\ 1})$.

$$\begin{aligned}\mathbf{S}[\![\lambda \mathtt{x.1}]\!] &= \Lambda R.\uparrow(\Lambda u.\mathsf{if}\ u \in \{\bot, \boldsymbol{\times}\}\ \mathsf{then}\ u\ \mathsf{else}\ \mathbf{S}[\![\mathtt{1}]\!]R[x \to u]) :: [\mathbb{U} \to \mathbb{U}]_\bot \\ &= \Lambda R.\uparrow(\Lambda u.\mathsf{if}\ u \in \{\bot, \boldsymbol{\times}\}\ \mathsf{then}\ u\ \mathsf{else}\ \uparrow(1) :: \mathbb{Z}_\bot) :: [\mathbb{U} \to \mathbb{U}]_\bot\end{aligned}$$

$$\mathbf{S}[\![\mathtt{if}\ (\lambda \mathtt{x.x})\ \mathtt{then\ 1\ else\ 1}]\!] = \Lambda R.\boldsymbol{\times} \tag{A.1}$$

$$\begin{aligned}\mathbf{S}[\![\phi]\!] &= \Lambda R.(\Lambda u.\mathsf{if}\ u \in \{\bot, \boldsymbol{\times}\}\ \mathsf{then}\ u\ \mathsf{else}\ \uparrow(1) :: \mathbb{Z}_\bot)(\boldsymbol{\times}) && \text{(A.2)} \\ &= \Lambda R.\boldsymbol{\times} && \text{(A.3)}\end{aligned}$$

$$\begin{aligned}\alpha^{\mathrm{Co}}(\mathbf{C}[\![\phi]\!]) &= \Lambda H.\bigwedge^{Co}\left\{t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \middle| \forall R \in \gamma_2^{\mathrm{Co}}(H) : \mathbf{S}[\![\phi]\!]R \in \gamma_1^{\mathrm{Co}}(t)\right\} \\ &= \Lambda H.\bigwedge^{Co}\left\{t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \middle| \forall R \in \gamma_2^{\mathrm{Co}}(H) : \boldsymbol{\times} \in \gamma_1^{\mathrm{Co}}(t)\right\} \\ &= \Lambda H.\bigwedge^{Co}\left\{\emptyset^{\mathrm{Co}}\right\} && \text{by Prop. A.1.2} \\ &= \Lambda H.\emptyset^{\mathrm{Co}} && \text{def. } \textstyle\bigwedge^{Co}\end{aligned}$$

$$\mathbf{T}^{\mathrm{Co}}[\![\lambda \mathtt{x}.1]\!] = \Lambda H. \bigwedge^{Co} \big\{ \{ t \to \mathbf{T}^{\mathrm{Co}}[\![1]\!] H[\mathtt{x} \leftarrow t] \} \big| t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \setminus \{\bot^{\mathrm{Co}}\} \big\} \quad (A.4)$$

$$= \Lambda H. \bigwedge^{Co} \big\{ \{ t \to \mathtt{int} \} \big| t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \setminus \{\bot^{\mathrm{Co}}\} \big\} \quad (A.5)$$

$$= \Lambda H. \big\{ t \to \mathtt{int} \big| t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \setminus \{\bot^{\mathrm{Co}}\} \big\} \quad (A.6)$$

$$\mathbf{T}^{\mathrm{Co}}[\![\mathtt{if}\ (\lambda \mathtt{x}.\mathtt{x})\ \mathtt{then}\ 1\ \mathtt{else}\ 1]\!] = \Lambda H. \emptyset^{\mathrm{Co}} \quad (A.7)$$

$$\mathbf{T}^{\mathrm{Co}}[\![\phi]\!] = \Lambda H. \bigwedge^{Co} \big\{ t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \big| \mathbf{T}^{\mathrm{Co}}[\![\lambda \mathtt{x}.1]\!] H \leq^{Co} \{ \mathbf{T}^{\mathrm{Co}}[\![\mathtt{if}\ (\lambda \mathtt{x}.\mathtt{x})\ \mathtt{then}\ 1\ \mathtt{else}\ 1]\!] H \to t \} \big\}$$
$$= \Lambda H. \bigwedge^{Co} \big\{ s \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \big| \big\{ t \to \mathtt{int} \big| t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \setminus \{\bot^{\mathrm{Co}}\} \big\} \leq^{Co} \{ \emptyset^{\mathrm{Co}} \to s \} \big\}$$

Looking at the definition of $\bigwedge^{Co}$, we can now easily see that, for each H, $\mathbf{T}^{\mathrm{Co}}[\![\phi]\!] H \leq^{Co} \mathtt{int} <^{Co} \emptyset^{\mathrm{Co}} = \alpha^{\mathrm{Co}}(\mathbf{C}[\![\phi]\!]) H$, contradicting Proposition 15. This follows from $\mathtt{int}$ being one of the $s \in \mathbb{P}^{\mathrm{Co}}_{\equiv}$ above, since $\gamma_1^{\mathrm{Co}}(\big\{ t \to \mathtt{int} \big| t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \setminus \{\bot^{\mathrm{Co}}\} \big\}) \subseteq \gamma_1^{\mathrm{Co}}(\{\emptyset^{\mathrm{Co}} \to \mathtt{int}\})$:

$$\begin{aligned}
&\gamma_1^{\mathrm{Co}}(\big\{ t \to \mathtt{int} \big| t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \setminus \{\bot^{\mathrm{Co}}\} \big\}) \\
&= \bigcap_{t \in \mathbb{P}^{\mathrm{Co}}_{\equiv}, t \neq \bot^{\mathrm{Co}}} \big\{ \psi \in [\mathbb{U} \to \mathbb{U}]_\bot \big| \forall u \in \gamma_1^{\mathrm{Co}}(t) : \psi(u) \in \gamma_1^{\mathrm{Co}}(\mathtt{int}) \big\} \cup \{\bot\} \\
&\subseteq \big\{ \psi \in [\mathbb{U} \to \mathbb{U}]_\bot \big| \forall u \in \gamma_1^{\mathrm{Co}}(\emptyset^{\mathrm{Co}}) : \psi(u) \in \gamma_1^{\mathrm{Co}}(\mathtt{int}) \big\} \cup \{\bot\} \\
&= \gamma_1^{\mathrm{Co}}(\{\emptyset^{\mathrm{Co}} \to \mathtt{int}\})
\end{aligned}$$

We now "derive" the proper rule. Let $H \in \mathbb{H}^{\mathrm{Co}}$ arbitrary.

$$\begin{aligned}
& \alpha^{\mathrm{Co}}(\mathbf{C}[\![\lambda \mathtt{x}.e]\!]) H \\
= \ & \bigwedge^{Co} \{ t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} | \forall R \in \gamma_2^{\mathrm{Co}}(H) : \mathbf{S}[\![\lambda \mathtt{x}.e]\!] R \in \gamma_1^{\mathrm{Co}}(t) \} && \text{prep., } (*) \\
= \ & \bigwedge^{Co} \{ t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} | \forall R \in \gamma_2^{\mathrm{Co}}(H) : \\
& \quad (\uparrow(\Lambda u.\mathsf{if}\ u \in \{\bot, \boldsymbol{\times}\}\ \mathsf{then}\ u\ \mathsf{else}\ \mathbf{S}[\![e]\!] R[x \leftarrow u]) :: [\mathbb{U} \to \mathbb{U}]_\bot) \in \gamma_1^{\mathrm{Co}}(t) \} && \text{def. } \mathbf{S}[\![\cdot]\!] \\
= \ & \bigwedge^{Co} \big(\{\emptyset^{\mathrm{Co}}\} \cup \{ T \subseteq \mathbb{P}^{\mathrm{Co}}_{\equiv} \times \mathbb{P}^{\mathrm{Co}}_{\equiv} | \forall R \in \gamma_2^{\mathrm{Co}}(H) : \forall t_1 \to t_2 \in T : \\
& \quad \big(\uparrow(\Lambda u.\mathsf{if}\ u \in \{\bot, \boldsymbol{\times}\}\ \mathsf{then}\ u\ \mathsf{else}\ \mathbf{S}[\![e]\!] R[x \leftarrow u]) :: [\mathbb{U} \to \mathbb{U}]_\bot\big) \in \vec{\gamma}^{\mathrm{Co}}(t_1 \to t_2) \} \big) && \text{def. } \gamma_1^{\mathrm{Co}} \\
& \text{But the } \emptyset^{\mathrm{Co}} \text{ case is subsumed by the polytypes through the empty polytype, hence} \\
= \ & \bigwedge^{Co} \{ T \subseteq \mathbb{P}^{\mathrm{Co}}_{\equiv} \times \mathbb{P}^{\mathrm{Co}}_{\equiv} | \forall R \in \gamma_2^{\mathrm{Co}}(H) : \forall t_1 \to t_2 \in T : \\
& \quad \big(\uparrow(\Lambda u.\mathsf{if}\ u \in \{\bot, \boldsymbol{\times}\}\ \mathsf{then}\ u\ \mathsf{else}\ \mathbf{S}[\![e]\!] R[x \leftarrow u]) :: [\mathbb{U} \to \mathbb{U}]_\bot\big) \in \vec{\gamma}^{\mathrm{Co}}(t_1 \to t_2) \} && \text{set identity} \\
= \ & \bigwedge^{Co} \{ T \subseteq \mathbb{P}^{\mathrm{Co}}_{\equiv} \times \mathbb{P}^{\mathrm{Co}}_{\equiv} | \forall R \in \gamma_2^{\mathrm{Co}}(H) : \forall t_1 \to t_2 \in T : \forall u \in \gamma_1^{\mathrm{Co}}(t_1) : && \text{def. } \vec{\gamma}^{\mathrm{Co}}, \\
& \quad u \neq \boldsymbol{\times} \wedge \mathbf{S}[\![e]\!] R[x \leftarrow u] \in \gamma_1^{\mathrm{Co}}(t_2) \\
& \quad \vee u = \boldsymbol{\times} \wedge u \in \gamma_1^{\mathrm{Co}}(t_2) \} && \text{Prop. A.1.1} \\
= \ & \bigwedge^{Co} \{ \{ t_1 \to t_2 \} | t_1 \to t_2 \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \times \mathbb{P}^{\mathrm{Co}}_{\equiv} \wedge \forall R \in \gamma_2^{\mathrm{Co}}(H) : \forall u \in \gamma_1^{\mathrm{Co}}(t_1) : \\
& \quad u \neq \boldsymbol{\times} \wedge \mathbf{S}[\![e]\!] R[x \leftarrow u] \in \gamma_1^{\mathrm{Co}}(t_2) \\
& \quad \vee u = \boldsymbol{\times} \wedge u \in \gamma_1^{\mathrm{Co}}(t_2) \} && \text{def. } \bigwedge^{Co} \\
\leq^{Co} \ & \bigwedge^{Co} \{ \{ t \to \mathbf{T}^{\mathrm{Co}}[\![e]\!] H[x \to t] \} | t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \wedge \forall R \in \gamma_2^{\mathrm{Co}}(H) : \forall u \in \gamma_1^{\mathrm{Co}}(t) :
\end{aligned}$$

$$
\begin{array}{lll}
& u \neq \text{✘} \wedge \mathbf{S}[\![e]\!]R[x \leftarrow u] \in \gamma_1^{\mathrm{Co}}(\mathbf{T}^{\mathrm{Co}}[\![e]\!]H[x \to t]) & \\
& \vee u = \text{✘} \wedge u \in \gamma_1^{\mathrm{Co}}(\mathbf{T}^{\mathrm{Co}}[\![e]\!]H[x \to t])\} & \text{def. } \leq^{Co} \\
\leq^{Co} & \bigwedge^{Co}\{\{t \to \mathbf{T}^{\mathrm{Co}}[\![e]\!]H[x \to t]\} | t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \setminus \{\emptyset^{\mathrm{Co}}\} \wedge \forall R \in \gamma_2^{\mathrm{Co}}(H) : \forall u \in \gamma_1^{\mathrm{Co}}(t) : & \\
& u \neq \text{✘} \wedge \mathbf{S}[\![e]\!]R[x \leftarrow u] \in \gamma_1^{\mathrm{Co}}(\mathbf{T}^{\mathrm{Co}}[\![e]\!]H[x \to t]) & \\
& \vee u = \text{✘} \wedge u \in \gamma_1^{\mathrm{Co}}(\mathbf{T}^{\mathrm{Co}}[\![e]\!]H[x \to t])\} & \text{def. } \leq^{Co} \\
= & \bigwedge^{Co}\{\{t \to \mathbf{T}^{\mathrm{Co}}[\![e]\!]H[x \to t]\} | t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \setminus \{\emptyset^{\mathrm{Co}}\} \wedge \forall R \in \gamma_2^{\mathrm{Co}}(H) : \forall u \in \gamma_1^{\mathrm{Co}}(t) : & \\
& \mathbf{S}[\![e]\!]R[x \leftarrow u] \in \gamma_1^{\mathrm{Co}}(\mathbf{T}^{\mathrm{Co}}[\![e]\!]H[x \to t])\} & \text{Prop. 2}
\end{array}
$$

But the latter is guaranteed by the induction hypothesis and Lemma 14.

$$
\begin{array}{lll}
= & \bigwedge^{Co}\{\{t \to \mathbf{T}^{\mathrm{Co}}[\![e]\!]H[x \to t]\} | t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \setminus \{\emptyset^{\mathrm{Co}}\}\} & \\
= & (\Lambda H. \bigwedge^{Co}\{\{t \to \mathbf{T}^{\mathrm{Co}}[\![e]\!]H[\mathtt{x} \leftarrow t]\} | t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \setminus \{\emptyset^{\mathrm{Co}}\}\})H & \eta\text{-conv.} \\
\neq & (\Lambda H. \bigwedge^{Co}\{\{t \to \mathbf{T}^{\mathrm{Co}}[\![e]\!]H[\mathtt{x} \leftarrow t]\} | t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \setminus \{\bot^{\mathrm{Co}}\}\})H & \\
= & \mathbf{T}^{\mathrm{Co}}[\![\lambda \mathtt{x}.e]\!]H & \text{def. } \mathbf{T}^{\mathrm{Co}}[\![\cdot]\!]
\end{array}
$$

Inconsistency 2: $\mathbf{T}^{\mathrm{Co}}[\![e_1 - e_2]\!]$ We might have used the minus operation to obtain a runtime error in the preceding example, but its typing rule is itself inconsistent. Consider $\phi = \mathtt{1} - \lambda \mathtt{x}.\mathtt{x}$, with arbitrary $H \in \mathbb{H}^{\mathrm{Co}}$:

$$
\begin{array}{lll}
& \alpha^{\mathrm{Co}}(\mathbf{C}[\![\mathtt{1} - \lambda \mathtt{x}.\mathtt{x}]\!])H & \\
= & \bigwedge^{Co}\left\{t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \middle| \forall R \in \gamma_2^{\mathrm{Co}}(H) : \mathbf{S}[\![\mathtt{1} - \lambda \mathtt{x}.\mathtt{x}]\!]R \in \gamma_1^{\mathrm{Co}}(t)\right\} & \\
= & \bigwedge^{Co}\left\{t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \middle| \forall R \in \gamma_2^{\mathrm{Co}}(H) : \text{✘} \in \gamma_1^{\mathrm{Co}}(t)\right\} & \\
= & \bigwedge^{Co}\left\{\emptyset^{\mathrm{Co}}\right\} & \text{Prop. A.1.2} \\
= & \bigcup\left\{\emptyset^{\mathrm{Co}}\right\} & \text{def. } \bigwedge^{Co} \\
= & \bigcup\{\emptyset\} & \text{def. } \emptyset^{\mathrm{Co}} \\
= & \emptyset & \text{def. } \bigcup \\
= & \emptyset^{\mathrm{Co}} & \text{def. } \emptyset^{\mathrm{Co}} \\
& >^{Co} \bot^{\mathrm{Co}} & \\
= & \mathtt{int} \wedge^{Co} \mathtt{int} \wedge^{Co} \left\{t \to t \middle| t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \setminus \{\bot^{\mathrm{Co}}\}\right\} & \\
= & \mathtt{int} \wedge^{Co} \mathbf{T}^{\mathrm{Co}}[\![\mathtt{1}]\!]H \wedge^{Co} \mathbf{T}^{\mathrm{Co}}[\![\lambda \mathtt{x}.\mathtt{x}]\!]H & \\
= & \mathbf{T}^{\mathrm{Co}}[\![\mathtt{1} - \lambda \mathtt{x}.\mathtt{x}]\!] &
\end{array}
$$

We conjecture that there has been some general downward/upward confusion and that the intended definition is in terms of lubs, not glbs, that is $\mathbf{T}^{\mathrm{Co}}[\![e_1 - e_2]\!] := \Lambda H.\mathtt{int} \vee^{Co} \mathbf{T}^{\mathrm{Co}}[\![e_1]\!]H \vee^{Co} \mathbf{T}^{\mathrm{Co}}[\![e_2]\!]H$. We prove the adequacy of this definition.

$$
\begin{array}{lll}
& \alpha^{\mathrm{Co}}(\mathbf{C}[\![e_1 - e_2]\!])H & \\
= & \bigwedge^{Co}\{t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} | \forall R \in \gamma_2^{\mathrm{Co}}(H) : \mathbf{S}[\![e_1 - e_2]\!]R \in \gamma_1^{\mathrm{Co}}(t)\} & \text{prep., } (*) \\
= & \bigwedge^{Co}\left\{t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} | \forall R \in \gamma_2^{\mathrm{Co}}(H) : \right. & \\
& \left. (\bot \in \{\mathbf{S}[\![e_1]\!]R, \mathbf{S}[\![e_2]\!]R\} \Rightarrow \bot \in \gamma_1^{\mathrm{Co}}(t)) \right. & \\
& \left. \wedge (\mathbf{S}[\![e_1]\!]R = z_1 :: \mathbb{Z}_\bot \wedge \mathbf{S}[\![e_2]\!]R = z_2 :: \mathbb{Z}_\bot \Rightarrow \uparrow(\downarrow(z_1) - \downarrow(z_2)) :: \mathbb{Z}_\bot \in \gamma_1^{\mathrm{Co}}(t)) \right. &
\end{array}
$$

$$
\begin{array}{lll}
& \wedge(\exists i \in \{1,2\} : \mathbf{S}[\![e_i]\!] \notin \mathbb{Z}_\perp \Rightarrow \text{✘} \in \gamma_1^{\mathrm{Co}}(t))\big\} & \text{def. } \mathbf{S}[\![\cdot]\!] \\
\leq^{Co} & \bigwedge^{Co} \big\{t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \big| \forall R \in \gamma_2^{\mathrm{Co}}(H) : \forall z \in \mathbb{Z}_\perp : z, \mathbf{S}[\![e_1]\!]R, \mathbf{S}[\![e_2]\!]R \in \gamma_1^{\mathrm{Co}}(t)\big\} & \text{def. } \bigwedge^{Co}\text{, Prop. A.1.5} \\
= & \bigwedge^{Co} \big\{t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \big| \mathtt{int}, \alpha^{\mathrm{Co}}(\mathbf{C}[\![e_1]\!])H, \alpha^{\mathrm{Co}}(\mathbf{C}[\![e_2]\!])H \leq^{Co} t\big\} & \text{Lemma 14} \\
\leq^{Co} & \bigwedge^{Co} \big\{t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \big| \mathtt{int}, \mathbf{T}^{\mathrm{Co}}[\![e_1]\!]H, \mathbf{T}^{\mathrm{Co}}[\![e_2]\!]H \leq^{Co} t\big\} & \text{def. } \leq^{Co}\text{, I.H.} \\
= & \bigwedge^{Co} \big\{t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \big| \gamma_1^{\mathrm{Co}}(\mathtt{int}), \gamma_1^{\mathrm{Co}}(\mathbf{T}^{\mathrm{Co}}[\![e_1]\!]H), \gamma_1^{\mathrm{Co}}(\mathbf{T}^{\mathrm{Co}}[\![e_2]\!]H) \subseteq \gamma_1^{\mathrm{Co}}(t)\big\} & \text{def. } \leq^{Co} \\
= & \bigwedge^{Co} \big\{t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \big| \bigcup\{\gamma_1^{\mathrm{Co}}(\mathtt{int}), \gamma_1^{\mathrm{Co}}(\mathbf{T}^{\mathrm{Co}}[\![e_1]\!]H), \gamma_1^{\mathrm{Co}}(\mathbf{T}^{\mathrm{Co}}[\![e_2]\!]H)\} \subseteq \gamma_1^{\mathrm{Co}}(t)\big\} & \text{def. } \bigcup \\
= & \alpha_1^{Co}(\bigcup\{\gamma_1^{\mathrm{Co}}(\mathtt{int}), \gamma_1^{\mathrm{Co}}(\mathbf{T}^{\mathrm{Co}}[\![e_1]\!]H), \gamma_1^{\mathrm{Co}}(\mathbf{T}^{\mathrm{Co}}[\![e_2]\!]H)\}) & \text{def. } \alpha_1^{\mathrm{Co}} \\
= & \bigvee^{Co}\{\mathtt{int}, \mathbf{T}^{\mathrm{Co}}[\![e_1]\!]H, \mathbf{T}^{\mathrm{Co}}[\![e_2]\!]H\} & \text{def. } \bigvee^{Co} \\
= & \mathtt{int} \vee^{Co} \mathbf{T}^{\mathrm{Co}}[\![e_1]\!]H \vee^{Co} \mathbf{T}^{\mathrm{Co}}[\![e_2]\!]H & \\
= & \mathbf{T}^{\mathrm{Co}}[\![e_1 - e_2]\!]H & \text{def. } \mathbf{T}^{\mathrm{Co}}[\![\cdot]\!]
\end{array}
$$

Proposition A.1.5. $\forall e_1, e_2 \in \mathbb{E} : \forall t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} : \forall H \in \mathbb{H}^{\mathrm{Co}} : \forall R \in \gamma_2^{\mathrm{Co}}(H) : \forall z \in \mathbb{Z}_\perp : z, \mathbf{S}[\![e_1]\!]R, \mathbf{S}[\![e_2]\!]R \in \gamma_1^{\mathrm{Co}}(t)$ *implies*

- $\perp \in \{\mathbf{S}[\![e_1]\!]R, \mathbf{S}[\![e_2]\!]R\} \Rightarrow \perp \in \gamma_1^{\mathrm{Co}}(t)$,
- $\mathbf{S}[\![e_1]\!]R = z_1 :: \mathbb{Z}_\perp \wedge \mathbf{S}[\![e_2]\!]R = z_2 :: \mathbb{Z}_\perp \Rightarrow \uparrow(\downarrow(z_1) - \downarrow(z_2)) :: \mathbb{Z}_\perp \in \gamma_1^{\mathrm{Co}}(t)$,
- $\exists i \in \{1,2\} : \mathbf{S}[\![e_i]\!] \notin \mathbb{Z}_\perp \Rightarrow \text{✘} \in \gamma_1^{\mathrm{Co}}(t)$.

We include the case of the zero test as an additional example.

$$
\begin{array}{lll}
& \alpha^{\mathrm{Co}}(\mathbf{C}[\![\mathtt{if}\ e_1\ \mathtt{then}\ e_2\ \mathtt{else}\ e_3]\!])H & \\
= & \bigwedge^{Co} \big\{t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \big| \forall R \in \gamma_2^{\mathrm{Co}}(H) : \mathbf{S}[\![\mathtt{if}\ e_1\ \mathtt{then}\ e_2\ \mathtt{else}\ e_3]\!]R \in \gamma_1^{\mathrm{Co}}(t)\big\} & \text{prep., } (*) \\
= & \bigwedge^{Co} \big\{t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \big| \forall R \in \gamma_2^{\mathrm{Co}}(H) : & \text{def. } \mathbf{S}[\![\cdot]\!] \\
& \quad (\mathbf{S}[\![e_1]\!]R = \perp \Rightarrow \perp \in \gamma_1^{\mathrm{Co}}(t)) & \\
& \wedge(\mathbf{S}[\![e_1]\!]R = \uparrow(0) :: \mathbb{Z}_\perp \Rightarrow \mathbf{S}[\![e_2]\!]R \in \gamma_1^{\mathrm{Co}}(t)) & \\
& \wedge(\mathbf{S}[\![e_1]\!]R = \uparrow(z) :: \mathbb{Z}_\perp \wedge z \neq 0 \Rightarrow \mathbf{S}[\![e_3]\!]R \in \gamma_1^{\mathrm{Co}}(t)) & \\
& \wedge(\mathbf{S}[\![e_1]\!]R \notin \mathbb{Z}_\perp \Rightarrow \text{✘} \in \gamma_1^{\mathrm{Co}}(t))\big\} & \\
= & \bigwedge^{Co} \big\{t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \big| \forall R \in \gamma_2^{\mathrm{Co}}(H) : & \text{def. } \gamma_1^{\mathrm{Co}} \\
& \quad (\mathbf{S}[\![e_1]\!]R = \perp \Rightarrow \perp \in \gamma_1^{\mathrm{Co}}(t)) & \\
& \wedge(\mathbf{S}[\![e_1]\!]R = \uparrow(0) :: \mathbb{Z}_\perp \Rightarrow \mathbf{S}[\![e_2]\!]R \in \gamma_1^{\mathrm{Co}}(t)) & \\
& \wedge(\mathbf{S}[\![e_1]\!]R = \uparrow(z) :: \mathbb{Z}_\perp \wedge z \neq 0 \Rightarrow \mathbf{S}[\![e_3]\!]R \in \gamma_1^{\mathrm{Co}}(t)) & \\
& \wedge(\mathbf{S}[\![e_1]\!]R \notin \gamma_1^{\mathrm{Co}}(\mathtt{int}) \Rightarrow \text{✘} \in \gamma_1^{\mathrm{Co}}(t))\big\} & \\
= & \bigwedge^{Co} \big\{t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \big| \forall R \in \gamma_2^{\mathrm{Co}}(H) : & \text{Lem. 14} \\
& \quad (\mathbf{S}[\![e_1]\!]R = \perp \Rightarrow \perp \in \gamma_1^{\mathrm{Co}}(t)) & \\
& \wedge(\mathbf{S}[\![e_1]\!]R = \uparrow(0) :: \mathbb{Z}_\perp \Rightarrow \mathbf{S}[\![e_2]\!]R \in \gamma_1^{\mathrm{Co}}(t)) & \\
& \wedge(\mathbf{S}[\![e_1]\!]R = \uparrow(z) :: \mathbb{Z}_\perp \wedge z \neq 0 \Rightarrow \mathbf{S}[\![e_3]\!]R \in \gamma_1^{\mathrm{Co}}(t)) & \\
& \wedge(\alpha^{\mathrm{Co}}(\mathbf{C}[\![e_1]\!])H \not\leq^{Co} \mathtt{int} \Rightarrow \text{✘} \in \gamma_1^{\mathrm{Co}}(t))\big\} & \\
= & \bigwedge^{Co} \big\{t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \big| \forall R \in \gamma_2^{\mathrm{Co}}(H) : & \text{by I.H.}
\end{array}
$$

$$
\begin{array}{lll}
 & (\mathbf{S}[\![e_1]\!]R = \bot \Rightarrow \bot \in \gamma_1^{\mathrm{Co}}(t)) & \\
 & \wedge(\mathbf{S}[\![e_1]\!]R = \uparrow(0) :: \mathbb{Z}_\bot \Rightarrow \mathbf{S}[\![e_2]\!]R \in \gamma_1^{\mathrm{Co}}(t)) & \\
 & \wedge(\mathbf{S}[\![e_1]\!]R = \uparrow(z) :: \mathbb{Z}_\bot \wedge z \neq 0 \Rightarrow \mathbf{S}[\![e_3]\!]R \in \gamma_1^{\mathrm{Co}}(t)) & \\
 & \wedge(\mathbf{T}^{\mathrm{Co}}[\![e_1]\!]H \neq \mathtt{int} \Rightarrow \boldsymbol{\times} \in \gamma_1^{\mathrm{Co}}(t))\big\} & \\
= & \bigwedge^{Co} \big\{t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \big| \forall R \in \gamma_2^{\mathrm{Co}}(H) : & \text{strengthening, Prop. A.1.2} \\
 & (\mathbf{S}[\![e_1]\!]R = \bot \Rightarrow \bot \in \gamma_1^{\mathrm{Co}}(t)) & \\
 & \wedge(\mathbf{S}[\![e_1]\!]R = \uparrow(0) :: \mathbb{Z}_\bot \Rightarrow \mathbf{S}[\![e_2]\!]R \in \gamma_1^{\mathrm{Co}}(t)) & \\
 & \wedge(\mathbf{S}[\![e_1]\!]R = \uparrow(z) :: \mathbb{Z}_\bot \wedge z \neq 0 \Rightarrow \mathbf{S}[\![e_3]\!]R \in \gamma_1^{\mathrm{Co}}(t)) & \\
 & \wedge(\mathbf{T}^{\mathrm{Co}}[\![e_1]\!]H \notin \{\bot^{\mathrm{Co}}, \mathtt{int}\} \Rightarrow t = \emptyset^{\mathrm{Co}})\big\} & \\
\leq^{Co} & \bigwedge^{Co} \big\{t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \big| \forall R \in \gamma_2^{\mathrm{Co}}(H) : & \text{contract} \\
 & (\mathbf{S}[\![e_1]\!]R = \bot \Rightarrow \bot \in \gamma_1^{\mathrm{Co}}(t)) & \\
 & \wedge(\mathbf{S}[\![e_1]\!]R \in \mathbb{Z}_\bot \Rightarrow \mathbf{S}[\![e_2]\!]R, \mathbf{S}[\![e_3]\!]R \in \gamma_1^{\mathrm{Co}}(t)) & \\
 & \wedge(\mathbf{T}^{\mathrm{Co}}[\![e_1]\!]H \notin \{\bot^{\mathrm{Co}}, \mathtt{int}\} \Rightarrow t = \emptyset^{\mathrm{Co}})\big\} & \\
= & \bigwedge^{Co} \big\{t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \big| \forall R \in \gamma_2^{\mathrm{Co}}(H) : & \text{Lem. 14, I.H.} \\
 & (\mathbf{S}[\![e_1]\!]R = \bot \Rightarrow \bot \in \gamma_1^{\mathrm{Co}}(t)) & \\
 & \wedge(\mathbf{S}[\![e_1]\!]R \in \mathbb{Z}_\bot \Rightarrow \mathbf{T}^{\mathrm{Co}}[\![e_2]\!]H, \mathbf{T}^{\mathrm{Co}}[\![e_3]\!]H \leq^{Co} t) & \\
 & \wedge(\mathbf{T}^{\mathrm{Co}}[\![e_1]\!]H \notin \{\bot^{\mathrm{Co}}, \mathtt{int}\} \Rightarrow t = \emptyset^{\mathrm{Co}})\big\} & \\
= & \bigwedge^{Co} \big\{t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \big| \forall R \in \gamma_2^{\mathrm{Co}}(H) : & \text{we're taking glb} \\
 & (\mathbf{S}[\![e_1]\!]R = \bot \Rightarrow \bot \in \gamma_1^{\mathrm{Co}}(t)) & \\
 & \wedge(\mathbf{S}[\![e_1]\!]R \in \mathbb{Z}_\bot \Rightarrow t = \mathbf{T}^{\mathrm{Co}}[\![e_2]\!]H \vee^{Co} \mathbf{T}^{\mathrm{Co}}[\![e_3]\!]H) & \\
 & \wedge(\mathbf{T}^{\mathrm{Co}}[\![e_1]\!]H \notin \{\bot^{\mathrm{Co}}, \mathtt{int}\} \Rightarrow t = \emptyset^{\mathrm{Co}})\big\} & \\
= & \bigwedge^{Co} \big\{t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \big| \forall R \in \gamma_2^{\mathrm{Co}}(H) : & \text{def. } \gamma_1^{\mathrm{Co}} \\
 & (\mathbf{S}[\![e_1]\!]R \in \gamma_1^{\mathrm{Co}}(\bot^{\mathrm{Co}}) \Rightarrow \gamma_1^{\mathrm{Co}}(t) = \gamma_1^{\mathrm{Co}}(\bot^{\mathrm{Co}})) & \\
 & \wedge(\mathbf{S}[\![e_1]\!]R \in \gamma_1^{\mathrm{Co}}(\mathtt{int}) \Rightarrow t = \mathbf{T}^{\mathrm{Co}}[\![e_2]\!]H \vee^{Co} \mathbf{T}^{\mathrm{Co}}[\![e_3]\!]H) & \\
 & \wedge(\mathbf{T}^{\mathrm{Co}}[\![e_1]\!]H \notin \{\bot^{\mathrm{Co}}, \mathtt{int}\} \Rightarrow t = \emptyset^{\mathrm{Co}})\big\} & \\
= & \bigwedge^{Co} \big\{t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \big| & \text{Lem. 14, def. } \leq^{Co} \\
 & (\alpha^{\mathrm{Co}}(\mathbf{C}[\![e_1]\!])H \leq^{Co} \bot^{\mathrm{Co}} \Rightarrow t = \bot^{\mathrm{Co}}) & \\
 & \wedge(\alpha^{\mathrm{Co}}(\mathbf{C}[\![e_1]\!])H \leq^{Co} \mathtt{int} \Rightarrow t = \mathbf{T}^{\mathrm{Co}}[\![e_2]\!]H \vee^{Co} \mathbf{T}^{\mathrm{Co}}[\![e_3]\!]H) & \\
 & \wedge(\mathbf{T}^{\mathrm{Co}}[\![e_1]\!]H \notin \{\bot^{\mathrm{Co}}, \mathtt{int}\} \Rightarrow t = \emptyset^{\mathrm{Co}})\big\} & \\
\leq^{Co} & \bigwedge^{Co} \big\{t \in \mathbb{P}^{\mathrm{Co}}_{\equiv} \big| & \text{I.H.} \\
 & (\mathbf{T}^{\mathrm{Co}}[\![e_1]\!]H = \bot^{\mathrm{Co}} \Rightarrow t = \bot^{\mathrm{Co}}) & \\
 & \wedge(\mathbf{T}^{\mathrm{Co}}[\![e_1]\!]H = \mathtt{int} \Rightarrow t = \mathbf{T}^{\mathrm{Co}}[\![e_2]\!]H \vee^{Co} \mathbf{T}^{\mathrm{Co}}[\![e_3]\!]H) & \\
 & \wedge(\mathbf{T}^{\mathrm{Co}}[\![e_1]\!]H \notin \{\bot^{\mathrm{Co}}, \mathtt{int}\} \Rightarrow t = \emptyset^{\mathrm{Co}})\big\} & \\
= & \mathsf{if}\ \mathbf{T}^{\mathrm{Co}}[\![e_1]\!]H = \bot^{\mathrm{Co}}\ \mathsf{then}\ \bot^{\mathrm{Co}} & \\
 & \mathsf{else\ if}\ \mathbf{T}^{\mathrm{Co}}[\![e_1]\!]H = \mathtt{int}\ \mathsf{then}\ \mathbf{T}^{\mathrm{Co}}[\![e_2]\!]H \vee^{Co} \mathbf{T}^{\mathrm{Co}}[\![e_3]\!]H & \\
 & \mathsf{else}\ \emptyset^{\mathrm{Co}} & \text{singleton } t
\end{array}
$$

$$= \quad \mathbf{T}^{\mathrm{Co}}[\![\texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3]\!]H \qquad \text{def. } \mathbf{T}^{\mathrm{Co}}[\![\cdot]\!]$$

Notation

arity Number of parameters in an overloading scheme. 26, 33, 34

C The set of type constructors with arities. 24, 25, 39, 41, 42

$\fallingdotseq$ A compatibility relation between identifiers and overloading assumptions. 25, 26, 34

constrain Computes substitution to alter a type term to match a variable constraint. 28, 29

cst Gets the constraints on type variables. 24, 27–29

dom Domain of a function. 25–28, 34

eval Evaluation of an identifier in an overloaded environment. 25, 26, 31, 33–35

$[u/x, v/y, \ldots]$ Finite map mapping x to u, y to v, etc..

FV Free type variables in type term. 24, 29, 30, 39

gfp Greatest fixed point. 8, 18

is-a Membership check for tagged values. 25, 26

lfp Least fixed point. 8, 16

$\mathbb{M}_{\vec{n}}^{PC}$ Set of all n-ary function types with constructors c_i. 35–38, 42

$\mathbb{M}_{\$}^{\mathrm{T}}$ Set of overloading schemes. 24

$\mathcal{P}$ The powerset, set of subsets. 11, 12, 14, 15, 18, 24, 38, 40

$\mathcal{P}_{fin}$ Set of finite subsets. 24

$\mathbb{P}_{\vec{n}}^{Co}$ Set of all n-ary function types with constructors c_i. 32, 34–37

resolve Creates unsaturated dispatcher for operator symbol. 26, 33, 34

$\xrightarrow{r}$ Used to denote respectful T-substitutions $S : \mathbb{V} \xrightarrow{r} T$. 28, 39–42

R_s Section of relation R by tuple s.

$<:$ Relates types to matching variable constraints. 27–29

$\mathcal{U}$ Computes respectful unifier for type terms. 29

$\mathbb{V}$ Type variables. 24, 27–29, 40–42, 56

ω Overloading scheme. 24–26, 30, 32–36, 39, 41

$\mathbb{X}$ Program variables. 7–9, 14, 15, 18, 22–27, 31, 33–35, 40, 49–53

Bibliography

[AEMO09] María Alpuente, Santiago Escobar, José Meseguer, and Pedro Ojeda, *Order-sorted generalization*, Electron. Notes Theor. Comput. Sci. **246** (2009), 27–38.

[AH87] S. Abramsky and C. Hankin, *Introduction to Abstract Interpretation*, Abstract Interpretation for Declarative Languages (S.Abramsky and C. Hankin, eds.), Ellis Horwood, 1987, pp. 9–31.

[BS01] F. Baader and W. Snyder, *Unification Theory*, Handbook of Automated Reasoning (J.A. Robinson and A. Voronkov, eds.), vol. I, Elsevier Science Publishers, 2001, pp. 447–533.

[Car96] Luca Cardelli, *Type Systems*, ACM Comput. Surv. **28** (1996), no. 1, 263–264.

[CC77] Patrick Cousot and Radhia Cousot, *Abstract Interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints*, Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (Los Angeles, California), ACM Press, New York, NY, 1977, pp. 238–252.

[CC14] Patrick Cousot and Radhia Cousot, *Abstract interpretation: Past, Present, and Future*, Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic In Computer Science (LICS) (Dale Miller and Thomas Henzinger, eds.), ACM, 2014, p. 12 p.

[CF58] Haskell B. Curry and Robert Feys, *Combinatory Logic, Volume I*, North-Holland, 1958.

[Chu40] Alonzo Church, *A Formulation of the Simple Theory of Types*, Journal of Symbolic Logic **5** (1940), no. 2, 56–68.

[Cou97] Patrick Cousot, *Types as Abstract Interpretations*, Conference Record of the Twentyfourth Annual ACM SIGPLAN-SIGACT

Symposium on Principles of Programming Languages (Paris, France), ACM Press, New York, NY, January 1997, pp. 316–331.

[Cou00] P. Cousot, *Abstract Interpretation: Achievements and Perspectives*, Proceedings of the SSGRR 2000 Computer & eBusiness International Conference (Compact disk paper 224 and electronic proceedings `http://www.ssgrr.it/en/ssgrr2000/proceedings.htm`, L'Aquila, Italy), Scuola Superiore G. Reiss Romoli, July 31 – August 6 2000.

[CW85] Luca Cardelli and Peter Wegner, *On Understanding Types, Data Abstraction, and Polymorphism*, ACM Comput. Surv. **17** (1985), no. 4, 471–523.

[DM82] Luis Damas and Robin Milner, *Principal Type-schemes for Functional Programs*, Proceedings of the 9th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (New York, NY, USA), POPL '82, ACM, 1982, pp. 207–212.

[DP02] Brian A. Davey and Hilary A. Priestley, *Introduction to Lattices and Order*, Cambridge University Press, 2002.

[GL02] Roberta Gori and Giorgio Levi, *An Experiment in Type Inference and Verification by Abstract Interpretation*, Verification, Model Checking, and Abstract Interpretation, Third International Workshop, VMCAI 2002, Venice, Italy, January 21-22, 2002, Revised Papers, 2002, pp. 225–239.

[GL03] ———, *Properties of a Type Abstract Interpreter*, Verification, Model Checking, and Abstract Interpretation, 4th International Conference, VMCAI 2003, New York, NY, USA, January 9-11, 2002, Proceedings, 2003, pp. 132–145.

[GL05] Ronald Garcia and Andrew Lumsdaine, *Type Classes Without Types*, Sixth Workshop on Scheme and Functional Programming (J. Michael Ashley and Michael Sperber, eds.), September 2005.

[Gra08] Jeremy Gray, *Plato's Ghost: The Modernist Transformation of Mathematics*, Princeton University Press, Princeton, NJ, 2008.

[GS90] C. A. Gunter and D. S. Scott, *Semantic Domains*, Handbook of Theoretical Computer Science (Vol. B) (Jan van Leeuwen, ed.), MIT Press, Cambridge, MA, USA, 1990, pp. 633–674.

[Han10] Stefan Hanenberg, *An Experiment About Static and Dynamic Type Systems: Doubts About the Positive Impact of Static Type Systems on Development Time*, Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages and Applications (New York, NY, USA), OOPSLA '10, ACM, 2010, pp. 22–35.

[HHJW07] Paul Hudak, John Hughes, Simon Peyton Jones, and Philip Wadler, *A History of Haskell: Being Lazy With class*, In Proceedings of the 3rd ACM SIGPLAN Conference on History of Programming Languages (HOPL-III), ACM Press, 2007, pp. 1–55.

[JJM97] Simon Peyton Jones, Mark Jones, and Erik Meijer, *Type Classes: An Exploration of the Design Space*, In Haskell Workshop, 1997.

[Jon92] Mark P. Jones, *A Theory of Qualified Types*, Proc. 4th European Symposium on Programming (ESOP), Rennes, France (Bernd Krieg-Brückner, ed.), Lecture Notes in Computer Science, vol. 582, Springer-Verlag, February 1992, pp. 287–306.

[Jon93] ———, *A System of Constructor Classes: Overloading and Implicit Higher-order Polymorphism*, Proceedings of the Conference on Functional Programming Languages and Computer Architecture (New York, NY, USA), FPCA '93, ACM, 1993, pp. 52–61.

[Jon95] ———, *Simplifying and Improving Qualified Types*, Proceedings of the Seventh International Conference on Functional Programming Languages and Computer Architecture (New York, NY, USA), FPCA '95, ACM, 1995, pp. 160–169.

[Kae88] Stefan Kaes, *Parametric Overloading in Polymorphic Programming Languages*, ESOP '88, 2nd European Symposium on Programming, Nancy, France, March 21-24, 1988, Proceedings (Harald Ganzinger, ed.), Lecture Notes in Computer Science, vol. 300, Springer, 1988, pp. 131–144.

[Kae05] ———, *Parametrischer Polymorphismus, Überladungen und Konversionen*, Ph.D. thesis, Technische Universität Darmstadt, 2005.

[Kli74] Morris Kline, *Why Johnny Can't Add: The Failure of the New Math*, Vintage Books, 1974.

[Mer15] *Polymorphism*, Merriam-Webster.com, 2015, `http://www.merriam-webster.com/dictionary/polymorphism`.

[MGS89] Jose Meseguer, Joseph A. Goguen, and Gert Smolka, *Order-sorted unification*, Journal of Symbolic Computation **8** (1989), no. 4, 383 – 413.

[Mil78] Robin Milner, *A Theory of Type Polymorphism in Programming*, Journal of Computer and System Sciences **17** (1978), 348–375.

[MPTN08] Radu Muschevici, Alex Potanin, Ewan Tempero, and James Noble, *Multiple Dispatch in Practice*, In: OOPSLA 08: Proceedings of the 23rd ACM International Conference on Object-Oriented Programming, Systems, Languages, and Applications, 2008, pp. 563–582.

[MTH90] Robin Milner, Mads Tofte, and Robert Harper, *Definition of Standard ML*, MIT Press, 1990.

[NNH99] Flemming Nielson, Hanne R. Nielson, and Chris Hankin, *Principles of Program Analysis*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1999.

[Odi99] Piergiorgio Odifreddi, *Classical Recursion Theory. Volume II*, Studies in Logic and the Foundations of Mathematics, North-Holland, Amsterdam, 1999.

[OSW97] Martin Odersky, Martin Sulzmann, and Martin Wehr, *Type Inference with Constrained Types*, Fourth International Workshop on Foundations of Object-Oriented Programming (FOOL), 1997.

[PLT14] PLT, *The Racket Language*, 2014, [Online; accessed 08/03/2014].

[PR05] Franois Pottier and Didier Rmy, *The Essence of ML Type Inference*, Advanced Topics in Types and Programming Languages (Benjamin C. Pierce, ed.), MIT Press, 2005, A draft extended version is also available, pp. 389–489.

[RCH12] Aseem Rastogi, Avik Chaudhuri, and Basil Hosmer, *The Ins and Outs of Gradual Type Inference*, Proceedings of the 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2012, Philadelphia, Pennsylvania, USA, January 22-28, 2012 (John Field and Michael Hicks, eds.), ACM, 2012, pp. 481–494.

[Ric53] H. G. Rice, *Classes of Recursively Enumerable Sets and Their Decision Problems*, Trans. Amer. Math. Soc. **74** (1953), 358–366.

[Rob65] J. A. Robinson, *A Machine-Oriented Logic Based on the Resolution Principle*, J. ACM **12** (1965), no. 1, 23–41.

[Sim14] Axel Simon, *Deriving a Complete Type Inference for Hindley-Milner and Vector Sizes using Expansion*, Science of Computer Programming (2014), preprint.

[Str00] Christopher Strachey, *Fundamental Concepts in Programming Languages*, Higher Order Symbol. Comput. **13** (2000), no. 1-2, 11–49.

[Tal13] David Tall, *How Humans Learn to Think Mathematically*, Cambridge University Press, 2013.

[THSAC+11] Sam Tobin-Hochstadt, Vincent St-Amour, Ryan Culpepper, Matthew Flatt, and Matthias Felleisen, *Languages As Libraries*, SIGPLAN Not. **46** (2011), no. 6, 132–141.

[VJSS11] Dimitrios Vytiniotis, Simon Peyton Jones, Tom Schrijvers, and Martin Sulzmann, *OutsideIn(X): Modular type inference with local assumptions*, Journal of Functional Programming (2011).

[WB89] P. Wadler and S. Blott, *How to Make Ad-hoc Polymorphism Less Ad Hoc*, Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (New York, NY, USA), POPL '89, ACM, 1989, pp. 60–76.