Type Inference with Bounded Quantification

Dilip Sequeira

Doctor of Philosophy
University of Edinburgh
1998

For my Parents

Abstract

In this thesis we study some of the problems which occur when type inference is used in a type system with subtyping. An underlying poset of atomic types is used as a basis for our subtyping systems. We argue that the class of *Helly* posets is of significant interest, as it includes lattices and trees, and is closed under type formation not only with structural constructors such as function space and list, but also records, tagged variants, Abadi-Cardelli object constructors, $\top$, and $\bot$. We develop a general theory relating consistency, solvability, and solution of sets of constraints between regular types built over Helly posets with these constructors, and introduce semantic notions of simplification and entailment for sets of constraints over Helly posets of base types. We extend Helly posets with inequalities of the form $\mathtt{a} \leqslant \tau$, where τ is not necessarily atomic, and show how this enables us to deal with *bounded quantification*.

Using bounded quantification we define a subtyping system which combines structural subtype polymorphism and predicative parametric polymorphism, and use this to extend with subtyping the type system of Laufer and Odersky for ML with type annotations. We define a complete algorithm which infers minimal types for our extension, using *factorisations*, solutions of subtyping problems analogous to principal unifiers for unification problems. We give some examples of typings computed by a prototype implementation.

Acknowledgements

I am grateful for the help of many people without whom I would not have been able to wrestle this thesis into submission.

Benjamin Pierce first engaged my interest in subtyping. I gained many useful insights from the comments and questions of Adriana Compagnoni, David Turner, and various members of the ML club of the Edinburgh LFCS. Professor Michael Fourman, whose considerable patience was fortunately not inexhaustible, provided useful advice in the construction of the dissertation. The examiners made valuable comments on an earlier version.

My parents were a source of constant understanding and boundless support. And Sophia, who drove me to many things, made all the difference in the world.

Declaration

I declare that this thesis was composed by myself and that the work contained therein is my own, except where explicitly stated otherwise in the text.

(*Dilip Sequeira*)

Table of Contents

Introduction

Subtyping

The study of *Subtyping* in the λ-calculus first arose in the work of Reynolds [45] and of Coppo and Dezani [10]. One type is a *subtype* of another if any element of the first may be used in any context where an element of the second is required; this is the principle of *subsumption*. We will write $\tau_1 \leqslant \tau_2$ to mean that τ_1 is a subtype of τ_2. Operationally this may denote either inclusion between types, or that an implicit coercion is to be inserted to transform an element of τ_1 to one of τ_2. Principal motivations for subtyping as a language construct are that it is

Natural to express semantic inclusion: for example in simple record type systems we expect a record with more fields to be usable wherever one with fewer fields is. And $\mathbb{N} \subseteq \mathbb{R}$, so we might intuitively expect `Nat` $\leqslant$ `Real`, even if at the machine level coercions may be required to implement this.

Powerful enough to encode the constructs of object-oriented languages within functional frameworks — see for example [37, 1], or [16, 14] for encodings in first-order systems of the kind we consider in this thesis.

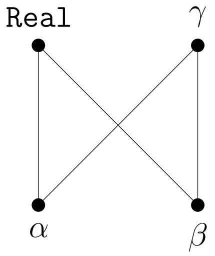
Type Inference

Realistic programming languages with complex type systems benefit greatly from type inference, the process of reconstructing type information omitted by the programmer. It is not without its drawbacks: the requirement that type inference be decidable places limits on the power of the type system, and interpreting the errors that arise during type inference generally requires more understanding of the type system by the programmer. But usually type inference speeds and simplifies coding tasks, and results in more elegant code.

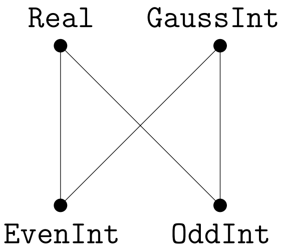
Although many notions of inference exist, our interest will be in versions of Hindley-Milner type inference [13] for λ-calculus extended with `let`-polymorphism, which is the basis for type inference in ML [27] and similar functional languages. Type inference for λ-calculus in the presence of subtyping was first considered by Mitchell [28]. It is significantly more complicated than without subtypes. Consider the following simple ML function:

```
fun max x y = if x < y then y else x
```

where `<` is a comparison of type $\texttt{Real}\rightarrow\texttt{Real}\rightarrow\texttt{Bool}$. Under the standard Core-ML type system, this program has type $\texttt{Real}\rightarrow\texttt{Real}\rightarrow\texttt{Real}$. However, suppose we have a subtype relation where `Real` has subtypes. Then if we assign the type variable α to `x` and β to `y`, type inference tells us that the term is typable for any α and β such that α and β are each subtypes of `Real` (from the `<`) and have a common supertype (so that the results of the two arms of the conditional can be combined.) The constraint graph is thus:



It is not unreasonable in general to want distinct values for α, β, and γ, all of which are different from `Real`. In the ML type system, all the inequalities are viewed as equalities and all points of this graph are identified. But in the presence of subtyping, identifying any pair may lead to a loss of solutions: suppose the underlying order is the (rather unrealistic) poset:



(Such a poset is called a *2-crown.*) We can use `max` to produce a `GaussInt` from an `EvenInt` and an `OddInt`, but it is easy to check that if we identify any two points in the constraint set for `max`, we will no longer be able to use the function in this way. Thus in general a principal type scheme will be a quantified type subject to constraints. We will write the principal type here as

$$\forall\{\alpha,\beta,\gamma\}\backslash\{\alpha\leqslant\texttt{Real},\beta\leqslant\texttt{Real},\alpha\leqslant\gamma,\beta\leqslant\gamma\}\ \alpha\rightarrow\beta\rightarrow\gamma$$

For such a simple expression this type is somewhat unwieldy, and it is shown in [22] that in general for any sound definition of an instance relation on constrained types there is a term of size k whose principal type scheme requires at least k constraints. More practically, the usual way of inferring types in the presence of subtyping yields a constraint for every application subterm, so that in typed λ-calculus the inferred constraint set grows at least linearly with the term; `let`-polymorphism can result in a much faster blowup. Thus we can expect a raw type inference algorithm to yield principal types with constraint sets which are too large to be understood by humans (e.g. quicksort produces approximately 300 constraints) and for inference to be practically scalable. Faced with constrained types, several issues arise immediately:

Satisfiability Is there a solution to the constraints? If there isn't, then we should generate a type error. In this particular instance it is easy to check by hand; in larger programs all but impossible.

Entailment Suppose we know what type we intended for this expression. Is it equivalent to, or at least less general than, the inferred type? This is the obvious question for signature matching, but it can also help during development: we can annotate program terms for documentary reasons, or, if a type error is encountered, to force the typechecker to check that it is producing the results intended at intermediate points. Entailment is a difficult problem, and not well-understood for complex type systems.

Simplification Is there a more succint representation for the constrained type? Generating small representations is important both for the typechecking algorithm, for reasons of scalability, and for the programmer, for comprehensibility.

The requirements of the type inference engine and the programmer in regard to simplification may be different: if the programmer is forced to look at types, it may be worth considerable effort to produce an optimally simplified solution. For purposes of efficient inference, however, a quickly generated but less than optimal simplification system may be more practical. In general, simplification strategies depend on the notion of entailment chosen, since this determines the classes of equivalent typings.

Solution Given a solvable constraint set, can we find a solution? Suppose we have a top-level function for which we infer type $\forall\overline{\alpha}\backslash C.\ \mathtt{Int}\rightarrow\mathtt{Int}$. None of the variables in C can have any effect outside the function, so the type scheme is unnecessarily polymorphic, and the actual instantiation of the variables $\overline{\alpha}$ is of no interest, save that the constraints in C are satisfied. Picking a particular solution enables us to compile the function monomorphically, which may represent a significant optimisation.

Types And Orderings

The question of what forms of subtyping can be feasibly implemented must be addressed in the light of how well we can deal with the above issues. In order to describe the relevant problems, we shall use some informal definitions here which will be made precise in chapter 1.

Subtype orders are typically constructed by asserting some ordering properties of the base types (`Nat`, `Int`, etc), and defining properties of the type constructors in the language (products, function types, lists, records etc) which propagate the subtyping relation on the base types. For example, we expect $\mathtt{Int}\to\mathtt{Nat} \leqslant \mathtt{Nat}\to\mathtt{Int}$, since the function space constructor is contravariant in its first argument, and covariant in its second. Constructors such as →, ×, and `List` are called *structural*: in an ordering built with such constructors from atomic types, any two elements which are in the subtype relation must have the same "shape", that is, lists of integers are only subtypes of lists of types of which integers are a subtype, etc. The record constructor is an example of a constructor that does not behave this way. Assume that we have simple records, in which our only operations are record formation and field extraction. Since all we can do to a record is extract a field, in any context where we need a record with a certain set of fields, a record with additional fields is also usable. This is known as "width-wise" subtyping: for example we have $\{a{:}\mathtt{Int}, b{:}\mathtt{Nat}\} \leqslant \{b{:}\mathtt{Int}\}$. This obviously doesn't preserve the same-shape property, and consequently such constructors are termed *non-structural*.

Whilst purely structural subtyping is easier to handle, in practice records and variants are important language features, as are the object constructors of [2], which have similar width-wise subtyping behaviour. One of the major issues of this thesis will be the question of how to combine existing techniques for structural subtyping to deal with non-structural constructors.

The question of whether the subtyping order may be allowed to vary is also an important one. It seems natural that if the language permits definition of new types, then it should be possible to create a place for them in the subtype ordering. This could either be by inclusion (a natural circumstance for subrange types), or by declaring an implicit coercion (for a continued fraction datatype to be coercible to a `Real`, say). We might also wish to define new types as subtypes of constructor types. For example, if we wished to implement complex numbers as (r, θ) pairs with $-\pi \leqslant \theta < \pi$, we might expect $\texttt{Complex} < \texttt{Real} \times \texttt{Real}$. More generally, an abstract type represented by a concrete type subject to some invariance condition may be seen as a subtype of the concrete representation.

Another reason for introducing such types is to allow *bounded quantification* [7], a language construct which allows universal quantifiers to quantify over only the subtypes of a given type, for example the $\forall\delta\leqslant\texttt{Real}.\ \delta\to\delta\to\delta$ typing for `max`. It is less expressive than full-blown constrained quantification, in that while all terms can be assigned principal constrained type schemes, not all terms have principal bounded type schemes. However, the question of when a constrained type scheme entails a bounded type scheme turns out to be quite straightforward to determine (whereas testing entailment between two constrained type schemes is not a well-understood problem).

In order to justify our approach to these issues, we will outline some of the results to date in the areas of satisfiability, entailment, and simplification.

Satisfiability

It is shown by Hoang and Mitchell in [22] that for any language which is based on the λ-calculus and incorporates subtyping, typability is PTIME-equivalent to satisfiability of constraint sets over the subtype ordering. Thus it is important to choose orderings for which satisfiability is tractable.

In [50], Tiuryn distinguishes between two forms of the satisfiability problem. The more general, UNIFORM-SSI, takes a constraint set and a poset as input, and checks satisfiability. The problem Q-SSI specialises UNIFORM-SSI to a particular poset. Tiuryn shows that there are posets (e.g. *crowns*, of which the 2-crown is the simplest) for which Q-SSI is PSPACE-hard, and thus so is UNIFORM-SSI (indeed it is PSPACE-complete [17]), but that for structural subtyping over unions of lattices it is in PTIME. This result was extended in [5] to Helly posets (a class which we will discuss further), and in [41] to a class of posets called *transitively feasible*. While it is not difficult to see how crowns might arise in order hierarchies (for example, those generated by multiple inheritance), in practice it is not feasible to incorporate them into subtyping systems.

Satisfiability is usually checked in polynomial time using some notion of consistency on the constraint set, rather than by constructing an explicit solution. Usually dynamic transitive closure of constraints is used: we close the constraint set under transitivity, and propagation of constraints of the form $\tau_1{\rightarrow}\tau_2 \leqslant \tau_1'{\rightarrow}\tau_2'$ to $\tau_1' \leqslant \tau_1$ and $\tau_2 \leqslant \tau_2'$. If this closure process terminates without generating inequalities between mismatching constructors, or constructors and atomic types, the constraint set is consistent and thus solvable. This technique requires the underlying poset to be a union of lattices. If subtyping is structural, then we can ensure this by choosing lattices for the posets of base types, since structural constructors always build lattices from lattices. But this is quite a strong restriction, since the natural orderings on types (for example numeric types) that we might wish to incorporate into a subtype hierarchy sometimes lack a natural choice of $\top$ type, and even more often a natural choice of $\perp$.

The method fails for trees even with structural subtyping: $\{\texttt{Nat}{\rightarrow}\texttt{Nat} \leqslant \alpha, \texttt{Long}{\rightarrow}\texttt{Long} \leqslant \alpha\}$ is unsolvable over the tree order $\texttt{Long} < \texttt{Int}, \texttt{Nat} < \texttt{Int}$, because any solution for α would have to be of the form $\texttt{a}_1{\rightarrow}\texttt{a}_2$ with the property that $\texttt{a}_1{\leqslant}\texttt{Long}$ and $\texttt{a}_1{\leqslant}\texttt{Int}$. Yet closure of the constraint set generates no inconsistencies.

Further, the set of records over a lattice (or even a discrete poset) does not form a lattice: for example, the types $\{l{:}\texttt{a}\}$ and $\{l{:}\{l{:}\texttt{a}\}\}$ possess an upper bound $\{\}$ but no lower bound. So constraint consistency is again inadequate: the constraint set $\{\{l{:}\texttt{a}\} \geqslant \alpha, \{l{:}\texttt{b}\} \geqslant \alpha\}$ is unsolvable even through constraint closure generates no inconsistencies. However, a modification of this method can be used with object types where field subtyping is invariant [32, 20]: in this case we need also to propagate common-lower-bound conditions to equalities between subterms: the corresponding constraint set $\{[l{:}\texttt{a}] \geqslant \alpha, [l{:}\texttt{b}] \geqslant \alpha\}$ using objects rather than records, yields the constraint $\texttt{a} = \texttt{b}$, demonstrating unsatisfiability.

In cases where records are required, the technique of [16] is to add global $\top$ and $\perp$ elements to the model of subtyping, making the ordering into one big lattice. Unfortunately, this can create undesirable side-effects: for example, consider the term

```
let f = fun(r) r.X + if r.X then 1 else 2;
```

The "one big lattice" technique accepts this term as well-typed: in fact it has the minimal type $\{X{:}\perp\}{\rightarrow}\texttt{Nat}$, since the `X` field of `r` is used at types `Bool` and `Int`. Unfortunately the domain of this type is uninhabited, so the function cannot be applied, which is unlikely to be what the programmer expected. If we solve constraint sets rather than just checking consistency, the presence of $\top$ or $\perp$ in an inferred type could be enough to alert the programmer to the fact that something is wrong, but the constraint may be propagated a long way through the program before it is detected. Or consider the term `true` $= 3$. If we have polymorphic equality, this term is typable, and in a PER model where all elements are equal at type $\top$, will always evaluate to `true`. This expression could quietly form part of a well-typed larger program, whereas a type error probably reflects the programmer's intuition more accurately.

These difficulties might be a fair price to pay in circumstances where $\top$ and $\perp$ are themselves useful, examples cited in [48] are for heterogeneous lists, or persistent data: weaker type systems that admit more judgements can also be seen as more flexible type systems. On the other hand, it is quite reasonable to want to add records, say, to a language, without the extra baggage resulting from $\top$ and $\perp$ So we are faced with the question: if the model is not then a lattice, what is it? In fact, the model containing just those types that are syntactically expressible turns out to have the structure of a Helly poset.

Once consistency has been checked, it may be possible (as in e.g. [16]) to demonstrate soundness of typing directly from the closure condition, but the usual method ([50, 5] for finite types; [34, 35] for regular infinite types) is to show that the consistency condition can be exploited to produce a solution.

Entailment

Semantically, one type scheme entails another if all solutions to the first are solutions to the second (e.g. [40, 52, 3]). For complex type systems no complete algorithm for testing entailment is known. In the context of subtyping over a lattice the entailment relation between a constraint set and a single constraint has been studied by Henglein and Rehof [21]. Our approach to entailment will be similar to theirs, but over a Helly poset rather than a lattice.

Simplification

Simplification techniques operate on the constraint set by using a sequence of transformations, each of which reduces its size. In general, the validity of a simplification transformation depends on the notion of equivalence between constrained types. For example, we have seen that in the inferred type of `max`, if we require that the set of solutions over a 2-crown be the same in any equivalent type, then no simplification is possible. On the other hand, if our underlying poset of types is a lattice, then in the case of `max`, α and β have a least upper bound δ, and in any solution this upper bound must be less than γ and `Real`. So any inputs of type α and β can be coerced to type δ, compared by coercing δ to `Real`, and an output of type δ can then be coerced to type γ. This process of "exporting" coercions leads us to an equivalent representation of the type of `max` as $\forall\delta\leqslant\texttt{Real}.\,\delta\rightarrow\delta\rightarrow\delta$.

There are two main identifiable streams of work in the literature. The simplest operates on constraints in atomic form. Introduced by Fuh and Mishra [18], its complexity and completeness properties have been more recently studied by Rehof [43]. The approach is essentially syntactic: a simplification being justified if it results in a provably equivalent constraint set.

The other operates on systems of constraints over regular types, often involving non-structural subtyping. The notion of equivalence here is semantic rather than syntactic, based on equality of solution sets. More simplifications are possible than in the syntactic approach, since information about the underlying poset (such as lattice structure) can be exploited. Early examples, such as [12, 46] were collections of transformations applied essentially heuristically. The most successful approach by Pottier [40] uses a powerful entailment algorithm as a foundation for the transformations, so that as well as transformations which known to be valid, the algorithm can speculatively transform the constraint set then test for equivalence, resulting in a very powerful simplification system. The incompleteness and high computational cost of the entailment algorithm are properties the simplification algorithm thus inherits. Simplification of constraint sets over regular types in a lattice of regular types over a discrete order extended with $\top$ and $\bot$ has been studied in [3], where a procedure is given which minimises the number of variables in a constraint set.

We will use our entailment relation over Helly posets to derive a simplification operation which allows us to exploit the Helly structure. For example, while in a Helly poset it is not generally the case that any pair of types possess a least upper bound, it is the case for any pair of types which have at least some upper bound. This suffices to demonstrate the existence of the type δ in the `max` example, and so justifies the same simplification as if the underlying subtype order were a lattice.

Type Inference and Annotation

The type system in which the subtype ordering is embedded is also an important consideration. Although for the most part we will consider the type system of ML extended with subtyping, called $\mathrm{ML}_{\leqslant}$, we shall also consider the system of Laufer and Odersky, which we shall call $\mathrm{ML}_{\forall}$. In core ML, all function arguments must have monomorphic types. Whilst this restriction is not impossible to lift [53], to do so without extending the term syntax results in the loss of principal typing. The most attractive approach, from [31], is to use *annotations*, whereby an abstraction is assumed to take a monomorphically typed argument unless the variable bound by the abstraction is annotated with a type scheme.

This Work

Helly Posets

We will concentrate on Helly posets, which we shall formally define in chapter 2. Helly posets possess attractive algebraic structure, and the relevance of this to efficient satisfiability was first demonstrated by Benke in [5]. The class of Helly posets is interesting for several reasons:

- It includes many classes of interesting posets, such as trees and lattices. Another example of a Helly poset is the Haskell class hierarchy [36].
- It is closed under orderings generated by many interesting type constructors.
- Satisfiability over Helly posets built with non-structural subtyping constructors is a smooth generalisation of unification and of transitive constraint closure.
- Helly posets can be extended with new elements in a natural fashion.

We will study entailment over Helly posets of atomic types, and indicate how it may be extended to the structural subtyping case, and we will introduce a simplification technique with a useful completeness property. We will also extend Benke's result to non-structural and non-finite cases, which will enable us to work with models of subtyping that do not have the problems which accompany $\top$ and $\bot$.

Non-atomic Constants

We will consider the impact of choosing a (not necessarily atomic) type τ in a Helly poset and adjoining a new type constant $\mathtt{a}$ with the inequality $\mathtt{a} \leqslant \tau$. This will enable us to introduce bounded quantification, and to determine whether a constrained type scheme entails a bounded type scheme with relatively low computational cost, by framing the entailment problem as a satisfiability problem. We will proceed to combine subtyping and annotations in a type system for which we will demonstrate an inference algorithm with a form of principal type.

Outline of the Thesis

The first part of the thesis will focus on the study of subtyping over Helly posets, the second on subtyping and annotation.

Preliminaries

Chapter 1 contains necessary background to the work.

Helly Posets

In chapter 2 we define Helly posets, and develop the theory which we shall apply in chapters 3 and 4.

Simplification and Entailment

In chapter 3 we define a simple entailment relation for atomic constraint sets over Helly posets, and show how it forms the foundation for a simplification transformation.

Satisfiability and Solution

In chapter 4 we show how constraint sets can be tested for satisfiability, and solved, over partial orders consisting of regular types built from Helly posets.

Subtyping and Annotations

In chapter 5 we investigate the relationship between bounded quantification and mixed-prefix unification. We define *quantified subtyping problems*, and show how such problems may be solved. In chapter 6 we show that partially solved problems have a factorisation property that is analogous to the principal solution property of first-order unification. We use this to incorporate subtyping in the extension of ML by annotations [31], and show that the resulting system has principal types (although not that it has principal typings).

Chapter 1

Preliminaries

In this chapter we will define some terms and formally state some result from the literature which we shall call on in later chapters.

1.1 Types

We will begin with a few definitions. The syntax of our types will be as follows:

$$\tau ::= \alpha \quad | \quad \mathtt{a} \quad | \quad T\tau_1 \ldots \tau_n$$

where T is a type constructor such as $\to$, `List` or a record type constructor, α ranges over type variables, and `a` ranges over base types. A *ground* type is a type containing no type variables. Each ground type is a *base* type if it is of the form `a`, and a *constructor* type otherwise. Base types can also be considered as constructors of arity zero. Our types will be built over some poset Q of *atoms*, whose elements cannot be coerced to elements of constructor types. Atoms are base types, but later we will allow assertions of the form $\mathtt{a} \leqslant \tau_1 \to \tau_2$ (for example), so that not all base types will necessarily be atoms. We will write $\tau(A)$ for the set of types built over some set A of base types.

1.2 Subtyping

We shall define the ordering over a poset Q of base types by a set of rules. Since it is an ordering, we naturally have

$$\frac{}{\tau \leqslant \tau} \qquad (\textsc{Taut})$$

$$\frac{\tau_1 \leqslant \tau_2 \qquad \tau_2 \leqslant \tau_3}{\tau_1 \leqslant \tau_3} \qquad (\textsc{Trans})$$

Additionally we need

- a finite description of the subtyping behaviour of the base types

$$\frac{\mathtt{a}{\leqslant}\mathtt{b} \in Q}{\mathtt{a} \leqslant \mathtt{b}} \qquad \text{(QAx)}$$

- a set of rules describing how subtyping between two constructor types is determined by the subtyping relationship between the constructor arguments, for example:

$$\frac{\tau_1' \leqslant \tau_1 \qquad \tau_2 \leqslant \tau_2'}{\tau_1{\rightarrow}\tau_2 \leqslant \tau_1'{\rightarrow}\tau_2'} \qquad \text{(Arrow)}$$

It will sometimes be convenient to consider the axioms for Q as constructor rules for zero-arity constructors.

Type Constructors

There are two classes of subtype orderings: *structural* and *non-structural*. Suppose we have an underlying poset Q of atomic types. There is a natural equivalence relation on Q, such that two elements are equivalent if they are in the same connected component (we shall use the term “component” to mean “connected component”). We will choose a representative for each equivalence class, and write $\ulcorner\mathtt{a}\urcorner$ for the representative of the class containing $\mathtt{a}$. Then we define the *shape* of a type as follows:

$$\begin{array}{lcl} \mathcal{S}(\mathtt{a}) & = & \ulcorner\mathtt{a}\urcorner \\ \mathcal{S}(T\tau_1\ldots\tau_n) & = & T\mathcal{S}(\tau_1)\ldots\mathcal{S}(\tau_n) \end{array}$$

A subtype ordering is *structural* if the components of its graph are exactly the sets of types of identical shape. A constructor will be called *structural* if extending a subtyping system with its constructor rule always preserves the structural subtyping property. For example, → is a structural constructor, while a global $\top$ type (such that $\tau \leqslant \top$ for every τ) is not, because it merges all the components of the ordering into one.

Since non-structural constructors have rather distinctive behaviours, we will deal with each on an individual basis. Structural constructors, however, will be dealt with uniformly. We use `T` to range over such constructors. We will use the generic term *labels* to indicate the set of indices used to order constructor arguments, so that → has labels 1 and 2. We use the function $L()$ to refer to the label set of a constructor, so that e.g. $L(\mathtt{List}) = \{1\}$, $L(\rightarrow) = \{1, 2\}$, $L(\mathtt{a}) = \varnothing$, and we extend this to types, so that $L(\tau)$ is the set of labels of the outermost constructor of τ.

Structural Constructors

Structural constructors include ×, +, `List`, and →. Each `T` has a fixed arity n, labels $\{1, \ldots n\}$, and an associated n-tuple of *polarities*, each of which is `pos`, `neg`, or `mix`, describing the subtyping behaviour of the constructor with respect to each argument. `pos` will mean that the constructor type increases as that argument does, `neg` that it decreases, and `mix` that changing that argument yields an incomparable type. For example the polarities of → would be $(\mathtt{neg}, \mathtt{pos})$, that of `ref`, $(\mathtt{mix})$. We will write $p^i_{\mathtt{T}}$ for the polarity of the constructor in its ith argument, and $\diamond^{\mathtt{T}}_i$ for whichever one of $\{\leqslant, \geqslant, =\}$ is required by the polarity on $\tau_i \leqslant \tau'_i$ for $\mathtt{T}\tau_1 \ldots \tau_1 \leqslant \mathtt{T}\tau'_1 \ldots \tau'_n$ to hold. Then for each structural constructor `T` we will have a rule:

$$\frac{\tau_i \diamond^{\mathtt{T}}_i \tau'_i \ \text{ for } 1 \leqslant i \leqslant n}{\mathtt{T}\tau_1 \ldots \tau_n \leqslant \mathtt{T}\tau'_1 \ldots \tau'_n} \qquad (\textsc{Cons-}\mathtt{T})$$

Given a type τ with outermost constructor `T` of arity n, we use the term $\tau(i)$ to denote the i^{th} constructor argument of `T`.

Records and Variants

Record subtyping is fundamental to encodings of objects in type systems such as that of [16]. The subtyping behaviour of the Abadi-Cardelli objects ([2]) is similar, and tagged variant types are dual to records in their subtyping behaviour and can be dealt with using the same techniques. The subtyping rule for records is:

$$\frac{\tau_i \leqslant \tau'_i \text{ for all } i \in I \qquad I \subseteq J}{\{l_j{:}\tau_j\}_{j \in J} \leqslant \{l_i{:}\tau'_i\}_{i \in I}} \qquad (\textsc{Record})$$

The record constructor can be considered of variable arity. For each finite set of labels I, we define a constructor $\{\}_I$ of arity $|I|$, which constructs a record with labels exactly the elements of I. Then $p^{\{\}_I}_l = \mathtt{pos}$ for any $l \in I$, and $L(\{\}_I) = I$. If τ is a record type, we use the term $\tau(l)$ to refer to the type of the l field.

$\top$ and $\bot$

$\top$ and $\bot$ are the other non-structural constructors usually encountered. Their subtyping rules are simply:

$$\frac{}{\tau \leqslant \top} \qquad (\textsc{Top})$$

$$\frac{}{\bot \leqslant \tau} \tag{Bot}$$

Obviously $L(\top) = L(\bot) = \varnothing$.

Nonatomic Base Types

We can extend the ordering over the underlying poset Q by a *bounded signature*. A bounded signature is a list of subtype assertions either of the form $\mathtt{a}$ where $\mathtt{a}$ does not occur previously in the signature or in Q, or $\mathtt{a}{\leqslant}\tau$, where $\mathtt{a}$ is a name not previously occurring in the signature or in Q, and τ is a type constructed using elements of Q and atoms previously bound in the signature.

$$\begin{array}{rcll} \Sigma & ::= & \bullet & \text{The empty signature} \\ & & \Sigma, \mathtt{a} & \\ & & \Sigma, \mathtt{a} \leqslant \tau & \text{where } \tau \in \tau(Q \cup \mathrm{atom}(\Sigma)) \end{array}$$

If $\mathtt{a}$ is bound in Σ, we will call it a Σ-atom (by contrast to a Q-atom), and for the set of such Σ-atoms we will write $\mathrm{atom}(\Sigma)$. It is straightforward that any extension of Q by a forest may be topologically sorted into a representation by a bounded signature. We have the rule $\textsc{Var}^{\uparrow}$:

$$\frac{\Sigma^{\uparrow}(\mathtt{a}) = \tau}{Q, \Sigma \ \vdash \ \mathtt{a} \leqslant \tau} \tag{$\textsc{Var}^{\uparrow}$}$$

where we write $\Sigma^{\uparrow}(\mathtt{a})$ for the upper bound of the Σ-atom $\mathtt{a}$. We will write $\Sigma(\mathtt{a}){\downarrow}$ to mean that $\mathtt{a}$ is a Σ-atom with an upper bound, and write $\Sigma^{\Uparrow}(\mathtt{a})$ for the minimal bound of Σ which is not a base type, if such exists.

It is easy to show that with just constructors and Q-atoms, transitivity is redundant: all occurrences of $\textsc{Trans}$ can be eliminated from any proof. However this is not the case once the rule $\textsc{Var}^{\uparrow}$ has been introduced. A standard technique [11, 38, 8] is to replace this rule with

$$\frac{\Sigma^{\uparrow}(\mathtt{a}) \leqslant \tau}{\mathtt{a} \leqslant \tau} \tag{$\textsc{Prom}^{\uparrow}$}$$

whence transitivity is again redundant.

Constraint Sets

A *constraint set* will be a set of assertions of the form $\tau_1 \leqslant \tau_2$ where τ_1 and τ_2 are types possibly containing variables. If C is a constraint set, a *solution* to C will be a ground substitution θ on the variables such that for every $\tau_1{\leqslant}\tau_2 \in C$, $\theta\tau_1 \leqslant \theta\tau_2$.

The set $\mathcal{O}(\tau)$ of *types occurring* in τ is defined as follows:

$$\begin{array}{lcl}
\mathcal{O}(\alpha) & = & \alpha \\
\mathcal{O}(\mathtt{a}) & = & \mathtt{a} \\
\mathcal{O}(c\tau_1 \ldots \tau_n) & = & \{c\tau_1 \ldots \tau_n\} \cup \mathcal{O}(\tau_1) \cup \ldots \cup \mathcal{O}(\tau_2)
\end{array}$$

If C is a constraint set, the set $\mathcal{O}(C)$ of types occurring in C is

$$\bigcup_{\tau_1 \leqslant \tau_2 \in C} \mathcal{O}(\tau_1) \cup \mathcal{O}(\tau_2)$$

1.3 Regular Types

Object systems which use subtyping typically require infinite types, and they are also a natural way to express types such as lists and trees. There are a number of ways in which to represent regular types, such as transition graphs ([25]), recursive types ([4, 6]), and trees ([4]), each of which has its advantages. We shall choose rooted transition graphs as our basic representation. A graph corresponding to a regular type is a finite directed graph with labelled vertices and edges, rooted at some particular vertex. The vertices are labelled with type constructors (via a labelling function ν), and a vertex labelled with the constructor c has exactly one out-edge for each $l \in L(c)$, and no others. We shall use (G, r) to range over such graphs, where r is the root vertex. For any vertex n in a transition graph with out-edge l, we shall write $n(l)$ for the vertex reached by an l-transition fron n. We shall use $\nu(\tau)$ to mean the outermost constructor of τ, which is just $\nu(r)$, where (G, r) is a graph of τ.

Any finite rooted graph can be unfolded from the root vertex into a regular tree (that is, a tree with finitely many distinct subtrees). We say two graphs (G_1, r_1) and (G_2, r_2) are *isomorphic* and write $(G_1, r_1) \cong (G_2, r_2)$ if they unfold to the same tree: that is, if every path in either graph exists in the other, and the path traverses the same sequence of constructors in each graph. For any given tree, there is an infinite isomorphism class of rooted graphs which unfold to it.

Shallow Approximation

It is convenient to extend the ordering on atoms (viewed as zero-arity constructors) to all constructors. To this end, we define the relation $\sqsubseteq$ as follows:

$$\begin{array}{lcll}
\mathtt{a} & \sqsubseteq & \mathtt{b} & \text{if } \mathtt{a} \leqslant \mathtt{b} \in Q \\
\rightarrow & \sqsubseteq & \rightarrow & \\
\{\}_{l \in L} & \sqsubseteq & \{\}_{m \in M} & \text{if } M \subseteq L \\
c & \sqsubseteq & \top & \text{for any constructor } c \\
\bot & \sqsubseteq & c & \text{for any constructor } c \\
\mathtt{a} & \sqsubseteq & c & \text{if } \Sigma^{\uparrow}(\mathtt{a}) \sqsubseteq c\text{, for non-atomic } \mathtt{a}
\end{array}$$

$\sqsubseteq$ on types is a "shallow" approximation of $\leqslant$: $\nu(\tau_1) \sqsubseteq \nu(\tau_2)$ is a necessary condition for $\tau_1 \leqslant \tau_2$, although clearly it is not sufficient.

1.3.1 Remark: Note that for all the constructors we consider, if $c_1 \sqsubseteq c_2$, then for any $l \in L(c_1) \cap L(c_2)$, $p_l^{c_1} = p_l^{c_2}$.

Simulations

1.3.2 Definition: A *simulation* between types built with the $\rightarrow$ constructor over some poset of base types is a relation $\lesssim$ on types such that if $(G_1, r_1) \lesssim (G_2, r_2)$

- $\nu(r_1) \sqsubseteq \nu(r_2)$
- if $\nu(r_1) = \nu(r_2) = \rightarrow$, then $(G_2, r_2(1)) \lesssim (G_1, r_1(1))$ and $(G_1, r_1(2)) \lesssim (G_2, r_2(2))$

It is easy to show that simulations are closed under unions, and it is well known (see e.g. [25]) that the $\leqslant$ relation is the largest simulation. Thus in order to demonstrate that $\tau_1 \leqslant \tau_2$, it suffices to find a simulation between them.

In fact, this result extends easily to other structural constructors, to records, $\top$, and $\bot$. We shall use a more general definition encompassing these constructors, although it can in fact be widened further, to deal with constructors for which remark(1.3.1) does not hold, e.g. updatable records [44].
$(G_1, r_1) \lesssim (G_2, r_2)$ is a simulation if

- $\nu(r_1) \sqsubseteq \nu(r_2)$
- if $l \in L(\nu(r_1)) \cap L(\nu(r_2))$,
  - if $p_l^{\nu(r_1)} = \texttt{pos}$, then $(G_1, r_1(l)) \lesssim (G_2, r_2(l))$
  - if $p_l^{\nu(r_1)} = \texttt{neg}$, then $(G_2, r_2(l)) \lesssim (G_1, r_1(l))$
  - if $p_l^{\nu(r_1)} = \texttt{mix}$, then $(G_1, r_1(l)) \lesssim (G_2, r_2(l))$ and $(G_2, r_2(l)) \lesssim (G_1, r_1(l))$

Notice that $(G_1, r_1) \cong (G_2, r_2)$ iff both $(G_1, r_1) \sqsubseteq (G_2, r_2)$ and $(G_2, r_2) \sqsubseteq (G_1, r_1)$. In fact, it is also straightforward to extend this approach to non-atomic base types also. For any $\texttt{a}$ occurring in Σ such that $\Sigma(\texttt{a})\!\downarrow$ let $(G_{\texttt{a}}, r_{\texttt{a}})$ be the graph of $\Sigma^{\uparrow}(\texttt{a})$. If $\tau \neq \texttt{a}$, then $\texttt{a} \leqslant \tau$ iff $\Sigma^{\Uparrow}(\texttt{a}) \leqslant \tau$, so it is sufficient to add to the above definition the condition

- if $\nu(r_1) = \texttt{a}$ such that $\Sigma(\texttt{a})\!\downarrow$, and $\nu(r_2) \neq \texttt{a}$, then $(G_{\texttt{a}}, r_{\texttt{a}}) \lesssim (G_2, r_2)$.

1.4 Type Systems and Inference

In this section we will outline some of the relatives of the ML typing system which we will discuss in later chapters.
Terms are

$$e \quad ::= \quad x \quad | \quad \lambda x.e \quad | \quad e_1 e_2 \quad | \quad \texttt{let}\ x = e_1\ \texttt{in}\ e_2$$

Types τ are as before, and type schemes κ have the form $\forall\overline{\alpha}\backslash C.\tau$, where C is a constraint set and $\overline{\alpha}$ is a set of type variables.

ML$_\leqslant$

In order to define ML$_\leqslant$, we alter our subtyping rules to have premises and conclusions of the form $C \vdash \tau_1 \leqslant \tau_2$. To the rules:

$$\frac{}{C \vdash \tau \leqslant \tau} \qquad \text{(Taut)}$$

$$\frac{C \vdash \tau_1 \leqslant \tau_2 \qquad C \vdash \tau_2 \leqslant \tau_3}{C \vdash \tau_1 \leqslant \tau_3} \qquad \text{(Trans)}$$

$$\frac{\mathtt{a} \leqslant \mathtt{b} \in Q}{C \vdash \mathtt{a} \leqslant \mathtt{b}} \qquad \text{(QAx)}$$

$$\frac{C \vdash \mathtt{b}_1 \leqslant \mathtt{a}_1 \qquad \mathtt{a}_2 \leqslant \mathtt{b}_2}{C \vdash \mathtt{a}_1{\rightarrow}\mathtt{a}_2 \leqslant \mathtt{b}_1{\rightarrow}\mathtt{b}_2} \qquad \text{(Arrow)}$$

we add the rule

$$\frac{\tau_1 \leqslant \tau_2 \in C}{C \vdash \tau_1 \leqslant \tau_2} \qquad \text{(Hyp)}$$

and write $C_1 \vdash C_2$ to mean that $C_1 \vdash \tau_1 \leqslant \tau_2$ for every $\tau_1 \leqslant \tau_2 \in C_2$. Note that transitivity can no longer be eliminated. We define the type system as follows:

$$\frac{\Gamma(x) = \kappa}{\Gamma; C \vdash x : \kappa} \qquad \text{(Var)}$$

$$\frac{\Gamma, x{:}\tau_1; C \vdash e : \tau_2}{\Gamma; C \vdash \lambda x.e : \tau_1{\rightarrow}\tau_2} \qquad \text{(Abs)}$$

$$\frac{\Gamma; C \vdash e_1 : \tau_1{\rightarrow}\tau_2 \qquad \Gamma; C \vdash e_2 : \tau_1}{\Gamma; C \vdash e_1 e_2 : \tau_2} \qquad \text{(App)}$$

$$\frac{\Gamma; C \vdash e_1 : \kappa_1 \qquad \Gamma, x{:}\kappa_1; C \vdash e_2 : \kappa_2}{\Gamma; C \vdash \texttt{let}\ x = e_1\ \texttt{in}\ e_2 : \kappa_2} \qquad \text{(Let)}$$

$$\frac{\Gamma; C \ \vdash \ e : \forall\overline{\alpha}\backslash C'.\tau \qquad C \ \vdash \ [\overline{\tau}/\overline{\alpha}]C'}{\Gamma; C \ \vdash \ e : [\overline{\tau}/\overline{\alpha}]\tau} \tag{\textsc{Inst}}$$

$$\frac{\Gamma; C \cup C_1 \ \vdash \ e : \tau \qquad \overline{\alpha} \cap (\mathrm{fv}(\Gamma) \cup \mathrm{fv}(C)) = \varnothing}{\Gamma; C \ \vdash \ \forall\overline{\alpha}\backslash C_1.\tau} \tag{\textsc{Gen}}$$

$$\frac{\Gamma; C \ \vdash \ e : \tau_1 \qquad C \ \vdash \ \tau_1 \leqslant \tau_2}{\Gamma; C \ \vdash \ e : \tau_2} \tag{\textsc{Sub}}$$

Type Inference

Type Inference in type systems with subtyping has two phases: extracting of a principal type and checking solvability of the resulting constraint set. As usual, we incorporate Inst and Gen into the other rules; Sub will be subsumed into the property of minimal typing that the algorithm possesses.

$$\frac{\Gamma(x) = \forall\overline{\alpha}\backslash C.\tau \qquad \overline{\beta} \text{ fresh}}{\Gamma; [\overline{\beta}/\overline{\alpha}]C \vdash_A \ x : [\overline{\beta}/\overline{\alpha}]\tau} \tag{A:\textsc{Var}}$$

$$\frac{\Gamma, x{:}\tau_1; C \vdash_A \ e : \tau_2}{\Gamma; C \vdash_A \ \lambda x.e : \tau_1 {\to} \tau_2} \tag{A:\textsc{Abs}}$$

$$\frac{\Gamma; C_1 \vdash_A \ e_1 : \tau_1 {\to} \tau_2 \qquad \Gamma; C_2 \vdash_A \ e_2 : \tau_1'}{\Gamma; C_1 \cup C_2 \cup \{\tau_1' \leqslant \tau_1\} \vdash_A \ e_1 e_2 : \tau_2} \tag{A:\textsc{App}}$$

$$\frac{\Gamma; C_1 \vdash_A \ e_1 : \tau \qquad \Gamma, x{:}\mathrm{close}(\Gamma, \tau, C_1); C_2 \ \vdash \ e_2 : \tau_2}{\Gamma; C_1 \cup C_2 \vdash_A \ \texttt{let} \ x = e_1 \ \texttt{in} \ e_2 : \tau_2} \tag{A:\textsc{Let}}$$

where $\mathrm{close}(\Gamma, \tau, C)$ is $\forall\overline{\alpha}\backslash C.\tau$ where $\overline{\alpha}$ is the set of variables occurring free in C and τ but not in Γ. It is easy to see that by inserting appropriate Inst and Gen rules we can rewrite an algorithmic derivation in the standard proof system in a canonical fashion. The constraint set obtained by this algorithm can then be tested for solvability as described earlier. These type inference rules have a principal typing property, for which we shall have to define an instance relation. The relation we use is Henglein's *halbstark* relation [19].

1.4.1 Definition: $\mathcal{D}_1 \equiv \Gamma_1; C_1 \ \vdash \ e : \tau_1$ has as an instance $\mathcal{D}_2 \equiv \Gamma_2; C_2 \ \vdash \ e : \tau_2$ (we write $\mathcal{D}_1 \ll \mathcal{D}_2$) if there is a substitution θ such that

1. $C_2 \ \vdash \ \theta(C) \cup \{\theta\tau_1 \leqslant \tau_2\}$

2. $\Gamma_2 = \theta(\Gamma_1)$

We extend this notion to type schemes as follows: Let $\overline{\alpha}_1$ not be free in C_1 and $\overline{\alpha}_2$ not free in C_2. Then $\Gamma_1; C_1 \ \vdash \ e : \forall\overline{\alpha}_1 \backslash D_1.\, \tau_1$ has as an instance $\Gamma_2; C_2 \ \vdash \ e : \forall\overline{\alpha}_2 \backslash D_2.\, \tau_2$ if

$$\Gamma; C_1 \cup D_1 \ \vdash \ e : \tau_1 \quad \ll \quad \Gamma; C_2 \cup D_2 \ \vdash \ e : \tau_2$$

Then we have the following (again from [19]):

1.4.2 Proposition: If $\Gamma; C \ \vdash \ e : \kappa$, then $\Gamma; C_A \vdash_A \ e : \tau_A$, and

$$\Gamma; C_A \vdash_A \ e : \tau_A \quad \ll \quad \Gamma; C \ \vdash \ e : \kappa_1$$

Thus every instance of a typing judgement is a halbstark instance of the type returned by the inference algorithm, and thus the inference algorithm provides a *minimal* type. In fact, we can also obtain all possible typing judgements for the term from the inferred typing using the rules Inst, Gen, Sub, and weakening on constraint sets, so the inferred typing is also *principal*.

ML$_\forall$

Another idea for extending ML from [31] is designed to allow passing of polymorphic functions as function arguments. This is done by means of *annotations*: λ-bound variables are assumed to have monomorphic type unless they are annotated with type schemes. Polymorphic type schemes are still impredicative: they may only be instantiated with monomorphic types. A subtype ordering is defined as follows:

$$\frac{}{\sigma \leqslant \sigma} \qquad \text{(REFL)}$$

$$\frac{\sigma_1 \leqslant \sigma_2 \qquad \sigma_2 \leqslant \sigma_3}{\sigma_1 \leqslant \sigma_3} \qquad \text{(Trans)}$$

$$\frac{\sigma_1' \leqslant \sigma_1 \qquad \sigma_2 \leqslant \sigma_2'}{\sigma_1 \rightarrow \sigma_2 \leqslant \sigma_1' \rightarrow \sigma_2'} \qquad \text{(Arrow)}$$

$$\frac{[\tau/\alpha]\sigma_1 \leqslant \sigma_2}{\forall\alpha.\sigma_1 \leqslant \sigma_2} \qquad (\forall\text{-LEFT})$$

$$\frac{\sigma_1 \leqslant \sigma_2 \qquad \alpha \text{ does not occur free in } \sigma_1}{\sigma_1 \leqslant \forall\alpha.\sigma_2} \qquad (\forall\text{-RIGHT})$$

Again there is a type inference algorithm with a minimal (and equivalently in this case principal) type property: if $\Gamma \vdash_A \ e : \sigma$ then for any σ' such that

$\Gamma \vdash e : \sigma'$, we have $\sigma \leqslant \sigma'$. In the inference algorithm, when a function accepting a parameter of type σ is applied to an argument with type σ', it is necessary to derive a principal substitution for σ' as an instance of σ; this is computed using *mixed prefix* unification [26]

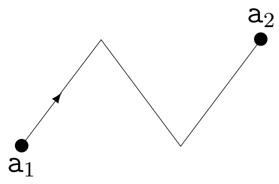
Chapter 2

Helly Posets

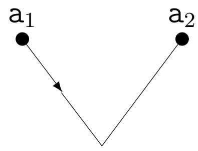
The study of partial orders underlies any theory of subtyping. Since our interest will be mainly in subtype orderings which possess the Helly property, in this chapter we will present some concepts and results which we shall apply to problems in subtyping in the rest of the thesis. The notion of distance in section(2.1) is from [42], the results on partial orders in section(2.2) are adapted from [30], and the results in section(2.3) are adapted from [5].

2.1 Fences and Distances

A *up-fence* of length n between two elements $\mathtt{a}_1$ and $\mathtt{a}_2$ of a poset is a sequence $\mathtt{a}_1 = \mathtt{b}_0 \leqslant \mathtt{b}_1 \geqslant \mathtt{b}_2 \ldots \mathtt{b}_n = \mathtt{a}_2$. A *down-fence* is defined dually.



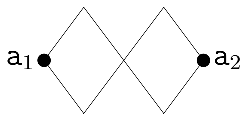
An up-fence of length 3



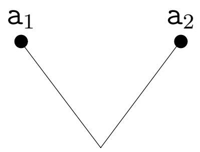
A down-fence of length 2

By convention, the fences from $\mathtt{a}_1$ to itself have length 1. The *distance* from one element to another is the pair $\langle u, d\rangle$ consisting of the lengths of the minimal up- and down-fences.

Notice that if we have an up-fence from $\mathtt{a}_1 = \mathtt{b}_0 \leqslant \mathtt{b}_1 \geqslant \ldots \mathtt{b}_n = \mathtt{a}_2$, we have a down-fence $\mathtt{a}_1 = \mathtt{a}_1 \geqslant \mathtt{b}_0 \leqslant \mathtt{b}_1 \geqslant \ldots \mathtt{b}_n = \mathtt{a}_2$ of length $n + 1$, thus in any distance $\langle u, d\rangle$, u and d differ by at most 1. Thus we can represent e.g. $\langle 3, 4\rangle$ graphically by just an up-fence of length 3, and e.g. $\langle 3, 2\rangle$ by just a down-fence of length 2; to represent $\langle 3, 3\rangle$ however, we need both fences.



$d(\mathtt{a}_1, \mathtt{a}_2) = \langle 3, 3 \rangle$



$d(\mathtt{a}_1, \mathtt{a}_2) = \langle 3, 2 \rangle$

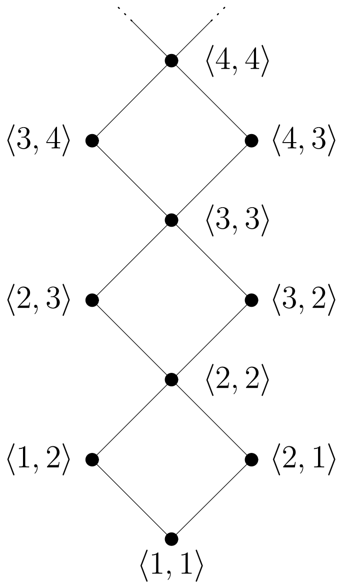
We will use e to range over distances, and write $d_Q(\mathtt{a}, \mathtt{b})$ for the distance from $\mathtt{a}$ to $\mathtt{b}$ in the poset Q; when Q is evident from context, we shall write simply $d(\mathtt{a}, \mathtt{b})$. The *diameter* δ_Q of a poset Q is the least e such that $d_Q(\mathtt{a}, \mathtt{b}) \leqslant e$ for all $\mathtt{a}, \mathtt{b}$ in the same component of Q.

Operations on distances

We will make extensive use of operations on distances. Here we outline the properties of these operations, and motivate the definitions of the less obvious ones.

Max and Min

With pointwise ordering, the distances form a lattice:

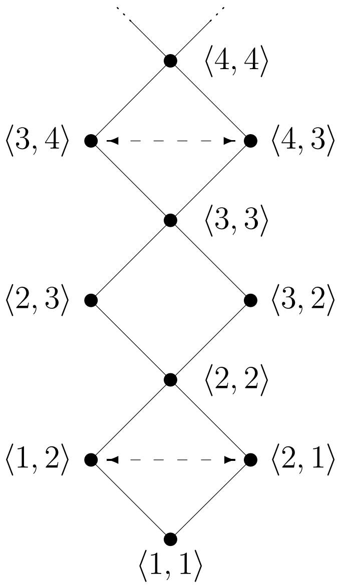


$$\mathcal{L} = \{\langle m, n \rangle \in \mathbb{Z}^+ \times \mathbb{Z}^+ \mid |m - n| \leqslant 1\} \cup \{\infty\}$$

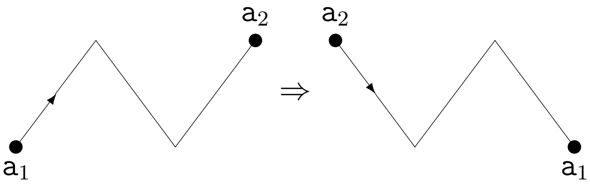
We will use $\vee$ and $\wedge$ for maximum and minimum respectively.

Reversal

If $d(\mathtt{a}_1, \mathtt{a}_2) = \langle u, d \rangle$, then the *reverse* $\langle u, d \rangle^{-1}$ is $d(\mathtt{a}_2, \mathtt{a}_1)$. Reversal flips fences horizontally, so the effect on $\mathcal{L}$ and on a particular distance can be visualised thus:



$e \mapsto e^{-1}$



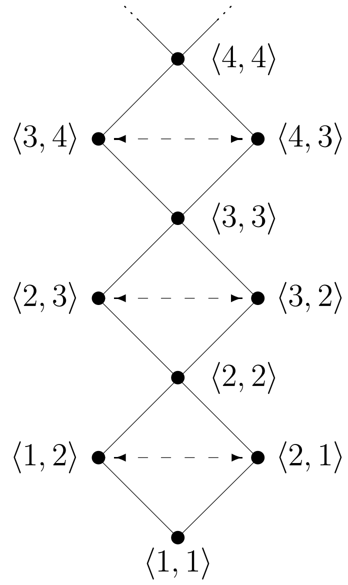
$\langle 3,4\rangle^{-1} = \langle 4,3\rangle$

Reversal is monotone, and

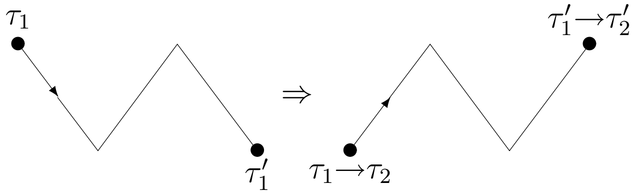
$$\begin{array}{lcl} (e^{-1})^{-1} & = & e \\ (e_1 \wedge e_2)^{-1} & = & (e_1^{-1} \wedge e_2^{-1}) \end{array}$$

Conjugation

The *conjugate* of $\langle u,d\rangle$ is $\langle u,d\rangle^* = \langle d,u\rangle$. Suppose we are dealing with structural subtyping over some poset of atomic base types. If we have a distance inequality $d(\tau_1{\rightarrow}\tau_2, \tau_1'{\rightarrow}\tau_2') \leqslant \langle 3,4\rangle$, then we know there are $\alpha_1{\rightarrow}\alpha_2$ and $\beta_1{\rightarrow}\beta_2$ such that $\tau_1{\rightarrow}\tau_2 \leqslant \alpha_1{\rightarrow}\alpha_2 \geqslant \beta_1{\rightarrow}\beta_2 \leqslant \tau_1'{\rightarrow}\tau_2'$. By contravariance therefore, we have $\tau_1 \geqslant \alpha_1 \leqslant \beta_1 \geqslant \tau_1'$. In other words, contravariance causes distances to be flipped vertically: up-fences are mapped to down-fences and vice versa. Conjugation represents this, and we can visualise its effect thus:



$e \mapsto e^*$



$\langle 4,3\rangle^* = \langle 3,4\rangle$

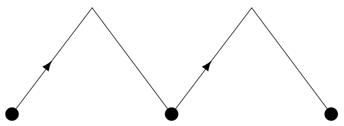
Conjugation is monotone, and

$$\begin{array}{lcl} (e^*)^* & = & e \\ (e^{-1})^* & = & (e^*)^{-1} \\ (e_1 \wedge e_2)^* & = & e_1^* \wedge e_2^* \end{array}$$

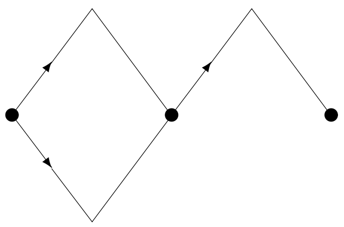
Observe that if $e_1 \nleqslant e_2$, although we do not have $e_2 \leqslant e_1$, we do have $e_2 \leqslant e_1^*$.

Addition

We define $e_1 + e_2$ as the minimum distance given by the concatenation of the fences forming e_1 and e_2.



$\langle 2,3\rangle + \langle 2,3\rangle = \langle 4,5\rangle$



$\langle 2,2\rangle + \langle 2,3\rangle = \langle 4,3\rangle$

Numerically, $\langle u_1, d_1\rangle + \langle u_2, d_2\rangle =$

$$\begin{array}{ll}
\langle u_1 + d_2 - 1, d_1 + u_2 - 1\rangle & \text{if } u_1 \text{ is even and } d_1 \text{ is even} \\
\langle u_1 + u_2 - 1, d_1 + u_2 - 1\rangle & \text{if } u_1 \text{ is odd and } d_1 \text{ is even} \\
\langle u_1 + d_2 - 1, d_1 + d_2 - 1\rangle & \text{if } u_1 \text{ is even and } d_1 \text{ is odd} \\
\langle u_1 + u_2 - 1, d_1 + d_2 - 1\rangle & \text{if } u_1 \text{ is odd and } d_1 \text{ is odd}
\end{array}$$

Addition is associative and monotone, and

$$\begin{array}{lcl}
(e_1 + e_2)^* & = & e_1^* + e_2^* \\
(e_1 + e_2)^{-1} & = & e_2^{-1} + e_1^{-1} \\
(e_1 \wedge e_2) + (e_3 \wedge e_4) & = & (e_1 + e_3) \ \wedge \ (e_2 + e_3) \ \wedge \ (e_1 + e_4) \ \wedge \ (e_2 + e_4)
\end{array}$$

Raising and Lowering

Suppose we know that the minimal up-fence from $\mathtt{a}$ to $\mathtt{b}$ is shorter than the minimal down-fence, for example $\mathtt{a} \neq \mathtt{b}$ and there is no element $\mathtt{c} < \mathtt{a}$. If we know also that $d(\mathtt{a}, \mathtt{b}) \leqslant \langle u, d\rangle$, we may deduce that $d(\mathtt{a}, \mathtt{b}) \leqslant \langle \max(d-1, 1), d\rangle$. Thus we define

$$\begin{array}{llcl}
\text{raising} & \langle u, d\rangle^{\vartriangle} & = & \langle \max(d-1,1), d\rangle \\
\text{lowering} & \langle u, d\rangle^{\triangledown} & = & \langle u, \max(u-1,1)\rangle
\end{array}$$

Raising and lowering are evidently monotone, and we additionally have:

$$\begin{array}{lcl}
(e_1 + e_2)^{\vartriangle} & = & e_1^{\vartriangle} + e_2 \\
(e_1 + e_2)^{\triangledown} & = & e_1^{\triangledown} + e_2
\end{array}$$

Dually, we define

$$\begin{array}{lcl}
e^{\blacktriangle} & = & ((e^{-1})^{\vartriangle})^{-1} \\
e^{\blacktriangledown} & = & ((e^{-1})^{\triangledown})^{-1}
\end{array}$$

so that if $\mathtt{a}$ is as above, and we know $d(\mathtt{b}, \mathtt{a}) \leqslant e$, then we can deduce $d(\mathtt{b}, \mathtt{a}) \leqslant d(\mathtt{b}, \mathtt{a})^{\blacktriangle}$. We have

$$\begin{array}{lcl}
(e_1 + e_2)^{\blacktriangle} & = & e_1 + e_2^{\blacktriangle} \\
(e_1 + e_2)^{\blacktriangledown} & = & e_1 + e_2^{\blacktriangledown}
\end{array}$$

2.2 Helly Posets

A *disc* $[\mathtt{a}, e]$ in a poset Q is the set of elements $\{\mathtt{b} \in Q \mid d(\mathtt{a}, \mathtt{b}) \leqslant e\}$. A set of discs $D = \{[\mathtt{a}_i, e_i]\}_{i \in I}$ constitute a *hole* if D has empty intersection, and no proper subset of D has empty intersection. If $P \supseteq Q$ is a poset, the hole D is *separated* in P if the set of discs D is also a hole in P — that is, there is no element of $P - Q$ contained in all the discs $\{[\mathtt{a}_i, e_i]\}_{i \in I}$

If P is a poset such that $Q \subseteq P$, a *retraction* from P to Q is an order-preserving map $f : P \to Q$ such that $f(q) = q$ for every $q \in Q$. If a retraction exists, we write $P \rhd Q$.

2.2.1 Proposition: If $Q \subseteq P$ and $P \rhd Q$, every hole in Q must be separated in P.

Proof: If $f : P \to Q$ is a retraction, it is straightforward by induction on the length of the fences that $d_Q(f\mathtt{a}, f\mathtt{b}) \leqslant d_P(\mathtt{a}, \mathtt{b})$. So suppose D is a hole in Q, and there is some element of $p \in P$ such that $p \in \bigcap D$. Then if $f : P \rightarrow Q$ were a retraction, for each disc $[\mathtt{a}, e] \in D$, $p \in [f\mathtt{a}, e]$, a contradiction to D being a hole in Q. □

Posets Q for which the separation of holes in $P \supseteq Q$ is sufficient as well as necessary for retractability are called *absolute retracts*.

2.2.2 Definition [Helly Posets]: A *Helly poset* Q is one in which the only holes are of size 2, i.e. for any set of discs D_i in Q, if

$$D_i \cap D_j \neq \varnothing \text{ for all } i \text{ and } j$$

then

$$\bigcap_i D_i \neq \varnothing$$

We have (from [30])

2.2.3 Proposition: Any Helly poset is an absolute retract.

That all holes have size two has the following consequence:

2.2.4 Lemma: Suppose $d_Q(\mathtt{a}_1, \mathtt{a}_2) \leqslant e$. Then for every e_1, e_2 such that $e_1 + e_2 \geqslant e$, there is some $\mathtt{b}$ such that $d(\mathtt{a}_1, \mathtt{b}) \leqslant e_1$ and $d_Q(\mathtt{b}, \mathtt{a}_2) \leqslant e_2$.

Proof: If $e_1 \geqslant e$ or $e_2 \geqslant e$, the result is trivial, so we assume otherwise and proceed by cases on the form of e.

- Suppose $e = \langle n, n+1 \rangle$.Then there is an up-fence F in Q of length n from $\mathtt{a}_1$ to $\mathtt{a}_2$. Let $\mathtt{b}$ be the most distant point from $\mathtt{a}_1$ on F such that $d_F(\mathtt{a}_1, \mathtt{b}) \leqslant e_1$. We must have $d_F(\mathtt{a}_1, \mathtt{b}) = e_1^{\vartriangle}$, since this is the distance along the maximal up-fence shorter than e.

  Now $e_1 + e_2 \geqslant e$, so $(e_1 + e_2)^{\vartriangle} \geqslant e^{\vartriangle}$, and since $e^{\vartriangle} = e$, $(e_1 + e_2)^{\vartriangle} \geqslant e$, and so $e_1^{\vartriangle} + e_2 \geqslant e$. Thus the distance e_2 is long enough to extend from $\mathtt{b}$ at least as far as $\mathtt{a}_2$. Thus we must have $d_Q(\mathtt{b}, \mathtt{a}_2) \leqslant e_2$. And since $d_F(\mathtt{a}_1, \mathtt{b}) \leqslant e_1$, $d_Q(\mathtt{a}_1, \mathtt{b}) \leqslant e_1$.

- If $e = \langle n+1, n\rangle$, a similar argument applies.
- Now suppose $e = \langle n, n\rangle$. If $e_1 + e_2 > e$, then either $e_1 + e_2 \geqslant \langle n, n+1\rangle$ or $e_1 + e_2 \geqslant \langle n+1, n\rangle$, and we can use one of the methods above to find $\mathtt{b}$. Otherwise we must have $e_1 = \langle m_1, m_1\rangle$, $e_2 = \langle m_2, m_2\rangle$, since only such a pair can sum to a distance of $\langle n, n\rangle$. Now by the argument above, the discs $[\mathtt{a}_1, \langle m_1, m_1 + 1\rangle]$ and $[\mathtt{a}_2, \langle m_2, m_2\rangle]$ must intersect, as must the discs $[\mathtt{a}_1, \langle m_1 + 1, m_1\rangle]$ and $[\mathtt{a}_2, \langle m_2, m_2\rangle]$. Since the discs $[\mathtt{a}_1, \langle m_1, m_1 + 1\rangle]$ and $[\mathtt{a}_2, \langle m_1 + 1, m_1\rangle]$ intersect trivially, all three discs must have a common intersection by the Helly property, and any point in this intersection suffices as a choice for $\mathtt{b}$. □

An immediate consequence of the above lemma is that two discs $[\mathtt{a}_1, e_1]$ and $[\mathtt{a}_2, e_2]$ intersect precisely if $d(\mathtt{a}_1, \mathtt{a}_2) \leqslant e_1 + e_2^{-1}$, i.e. if the distance between their centres is less than or equal to the sum of their radii

2.2.5 Proposition: If Q is a Helly poset and $Q \subseteq P$, then $P \rhd Q$ iff for every $\mathtt{a}_1, \mathtt{a}_2 \in Q$, $d_Q(\mathtt{a}_1, \mathtt{a}_2) \leqslant d_P(\mathtt{a}_1, \mathtt{a}_2)$.

Proof: Suppose $P \rhd Q$, then every fence in P retracts to a fence in Q, so $d_Q(\mathtt{a}_1, \mathtt{a}_2) \leqslant d_P(\mathtt{a}_1, \mathtt{a}_2)$. Conversely, suppose $P \not\rhd Q$. Then there is some hole $\{[\mathtt{a}_1, e_1], [\mathtt{a}_2, e_2]\}$ in Q which is not separated in P. So there is some element $\mathtt{b} \in P$ such that $d_P(\mathtt{a}_1, \mathtt{b}) \leqslant e_1$ and $d_P(\mathtt{a}_2, \mathtt{b}) \leqslant e_2$, so we have

$$d_P(\mathtt{a}_1, \mathtt{a}_2) \leqslant e_1 + e_2^{-1}$$

but we cannot have

$$d_Q(\mathtt{a}_1, \mathtt{a}_2) \leqslant e_1 + e_2^{-1}$$

since if we did, by lemma(2.2.4) there would be an element $\mathtt{c}$ such that $d_Q(\mathtt{a}_1, \mathtt{c}) \leqslant e_1$ and $d_Q(\mathtt{c}, \mathtt{a}_2) \leqslant e_2$, which would be a contradiction to the pair of discs forming a hole in Q □

Thus if Q is Helly, and $P \supseteq Q$, $P \rhd Q$ if all the holes of Q of size 2 are separated in P. If such a P is also Helly, we will say P is a *Q-model.*

2.2.6 Proposition: Suppose Q is Helly. P is a Q-model iff for every pair $\mathtt{a}_1, \mathtt{a}_2 \in Q$, $d_Q(\mathtt{a}_1, \mathtt{a}_2) = d_P(\mathtt{a}_1, \mathtt{a}_2)$.

Proof: Suppose P is a Q-model. Then every fence in Q also exists in P, so $d_P(\mathtt{a}_1, \mathtt{a}_2) \leqslant d_Q(\mathtt{a}_1, \mathtt{a}_2)$. And since $P \rhd Q$, $d_Q(\mathtt{a}_1, \mathtt{a}_2) \leqslant d_P(\mathtt{a}_1, \mathtt{a}_2)$. Conversely, if $d_Q(\mathtt{a}_1, \mathtt{a}_2) = d_P(\mathtt{a}_1, \mathtt{a}_2)$, $P \rhd Q$ by the Helly property for Q. □

Helly posets may also be characterised in terms of irreducible elements. An *upper cover* for an element $\mathtt{a}$ in a poset is an element $\mathtt{b}$ such that $\mathtt{a} < \mathtt{b}$, and there is no element $\mathtt{c}$ such that $\mathtt{a} < \mathtt{c} < \mathtt{b}$ (*lower cover* being defined dually). An element is *doubly irreducible* if it has at most one upper cover and one lower cover. A *decomposition by doubly irreducibles* of a finite poset Q is a sequence $\mathtt{q}_1, \mathtt{q}_2 \ldots$ containing all the elements of Q such that if $Q_1 = Q$, $Q_{i+1} = Q_i \backslash \mathtt{q}_i$, then each $\mathtt{q}_i$ is doubly irreducible in Q_i. Again from [30] we have:

2.2.7 Proposition: If Q is finite, Q is a Helly poset iff it is decomposable by doubly irreducibles.

An easy method of demonstrating that a finite poset is Helly is to adduce such a decomposition, which is immediate in the case of fences and trees, for example. It was stated in the introduction that Helly posets have the following property:

2.2.8 Proposition: In a Helly poset, any set of elements with an upper bound possess a least upper bound, and any set of elements with a lower bound possess a greatest lower bound.

Proof: We will do the first, the second is dual. Suppose $\{\mathtt{a}_i\}$ have a common upper bound: then the intersection of the discs $\{[\mathtt{a}_i, \langle 1, 2\rangle]\}$ is non-empty: suppose it is the set $\{\mathtt{b}_j\}$. Then for each j, $[\mathtt{b}_j, \langle 2, 1\rangle]$ intersects each disc $[\mathtt{a}_i, \langle 1, 2\rangle]$ (since each disc contains both points), and for j_1 and j_2, the discs $[\mathtt{b}_{j_1}, \langle 2, 1\rangle]$ and $[\mathtt{b}_{j_2}, \langle 2, 1\rangle]$ intersect at each $\mathtt{a}_i$. Thus each pair in the set $\{[\mathtt{a}_i, \langle 1, 2\rangle]\} \cup \{[\mathtt{b}_j, \langle 2, 1\rangle]\}$ intersects, and thus the whole set of discs have non-empty intersection. If $\mathtt{b}$ is contained in all of the discs, it must be an upper bound of $\{\mathtt{a}_i\}$ and since it is less than each $\mathtt{b}_i$, it must be the least such. □

Examples

Although not the largest class for which constraint solution can be tested in polynomial time, the class of Helly posets is attractive because its algebraic character enables us to decompose conditions involving a set of types and type variables into a set of easily checked conditions on the set of pairs of types. It has some simple closure properties: it is closed under products, retractions (so in fact it is a *variety*) and also disjoint unions and inversion. As remarked in the introduction, it is also closed under record type formation, we will show the case for finite types.

2.2.9 Lemma: Let Q^* be the poset of types formed over a Helly poset Q using structural and record type constructors. Any set of types τ_i in Q^* with pairwise upper bounds have a least upper bound, and any set of types τ_i with pairwise lower bounds possess a greater lower bound.

Proof: We will proceed by induction on the maximal depth of constructor nesting of any type in the set $\{\tau_i\}$. If the level of constructor nesting is 0, all the τ_i are base types, and the result is just proposition(2.2.8). Otherwise if one of the τ_i is a constructor type, say $\rightarrow$, all must be $\rightarrow$-types, and it is straightforward that

$$\bigvee_i \tau_i{\rightarrow}\tau'_i = \Big(\bigwedge_i \tau_i\Big) \rightarrow \Big(\bigvee_i \tau'_i\Big)$$

and dually for greatest lower bound. Other constructors are similar.

Similarly if one of the τ_i is a record type, all must be. Suppose not all the τ_i have an l-field, then no upper bound can have an l-field. And if all the τ_i have an l-field, there are two possible cases: either the set $\{\tau_i(l)\}$ has an upper bound, in which case by the induction hypothesis it has a least upper bound τ_l, so for any upper bound τ such that $\tau(l)$ is defined, we must have $\tau(l) \geqslant \tau_i(l)$ for each l, and so $\tau(l) \geqslant \tau_l$. Or the set has no upper bound, in which case no upper bound for the $\{\tau_i\}$ can have an l-field. Thus the least upper bound of the $\{\tau_i\}$ is

$$\bigvee_i \tau_i = \Big\{l : \bigvee_i \tau_i(l)\Big\}_{l \in J}$$

where J is the set of labels l occurring in every τ_i such that $\bigvee_i \tau_i(l)$ is defined. And suppose $\{\tau_i\}$ has a lower bound τ, then τ must have all the fields occurring in any τ_i, and $\tau(l) \leqslant \tau_i(l)$ for any i such that $l \in L(\tau_i)$. So $\tau(l)$ is a lower bound for the set $\{\tau_i(l) \mid l \in L(\tau_i)\}$, and so by the induction hypothesis, $\{\tau_i(l) \mid l \in L(\tau_i)\}$ has greatest lower bound. So the greatest lower bound of the $\{\tau_i\}$ is

$$\bigwedge_i \tau_i = \Big\{l : \bigwedge_{l \in L(\tau_i)} \tau_i(l)\Big\}_{l \in J}$$

where $J = \bigcup_i L(\tau_i)$. □

2.2.10 Proposition: The poset of types formed over a Helly poset using structural and record type constructors is a Helly poset

Proof: We will demonstrate that any set of discs $\mathcal{D} = [\tau_i, e_i]_{i \in I}$ in the poset which have pairwise intersection contain a common point. Again we will proceed by induction on the maximal depth of constructor nesting in any type on which a disc is centred.

From the subtyping rules, the τ_i must have the same outermost constructor. If this is anything other than a record constructor, the result follows easily from the induction hypothesis. So suppose the τ_i are all records. We may assume that no disc has radius $\langle 1, 1\rangle$ for then the result is trivial. We define

$$\rho = \bigwedge\{\tau \mid [\tau, \langle 2, 1\rangle] \in \mathcal{D}\}$$

which exists, because the discs $\{[\tau, \langle 1, 2\rangle]\}$ all intersect pairwise, so the types τ all have pairwise lower bounds, and thus by lemma(2.2.9) they have a common lower bound. It suffices to show that each disc in $\mathcal{D}$ contains ρ.

Since any set of records has the empty record $\{\}$ as an upper bound, any disc $[\tau, \langle u, d\rangle]$ with $u > 1$ and $d > 2$ is the entire set of record types, and thus must contain ρ. Notice also that any disc $[\tau, \langle 3, 2\rangle]$ contains precisely the same elements as the disc $[\tau, \langle 2, 2\rangle]$, since there is an up-fence of length 2 from any record to any other (via the empty record). Thus we need consider only the discs of radius $\langle 3, 2\rangle$ and $\langle 1, 2\rangle$.

Suppose $[\tau, \langle 1, 2\rangle] \in \mathcal{D}$. This disc intersects each $[\tau', \langle 2, 1\rangle] \in \mathcal{D}$, so

$$d(\tau, \tau') \leqslant \langle 1, 2\rangle + \langle 1, 2\rangle = \langle 1, 2\rangle$$

Thus τ is a lower bound for all such τ', and thus $\tau \leqslant \rho$, and so $\rho \in [\tau, \langle 1, 2\rangle]$.

And each $[\tau, \langle 3, 2\rangle] \in \mathcal{D}$ intersects each $[\tau', \langle 2, 1\rangle] \in \mathcal{D}$ so we have

$$d(\tau, \tau') \leqslant \langle 3, 2\rangle + \langle 1, 2\rangle = \langle 3, 2\rangle$$

Thus τ has a pairwise lower bound with each τ', and since each pair of τ' also have a pairwise lower bound, by lemma(2.2.9) the set $\{\tau'\}$ and the element τ have some greatest lower bound τ''. But we have $\tau'' \leqslant \rho$, since ρ is the greatest lower bound of just the τ'. Thus

$$\begin{array}{rcl} d(\tau, \rho) & \leqslant & d(\tau, \tau'') + d(\tau'', \rho) \\ & \leqslant & \langle 2, 1\rangle + \langle 1, 2\rangle \\ & \leqslant & \langle 3, 2\rangle \end{array}$$

Thus $\rho \in [\tau, \langle 3, 2\rangle]$. □

A similar result is that if a subtyping system with the Helly property is augmented with either or both of the constructors $\top$ and $\bot$, the Helly property is preserved.

2.3 Distance Inequalities

A *distance inequality* is an assertion of the form $d(\tau_1, \tau_2) \leqslant e$. Distance inequalities generalise simple inequalities: any constraint $\tau_1 \leqslant \tau_2$ can be expressed by the distance inequality $d(\tau_1, \tau_2) \leqslant \langle 1, 2\rangle$, and equally any distance inequality

$d(\tau_1, \tau_2) \leqslant e$ can be expressed as a set of constraints with fresh variables forming the intermediate points in the up- and down-fences.

Given a set of distance inequalities $\mathcal{I}$, we can deduce other inequalities using reversal and addition, and the fact that $\mathcal{L}$ has a lattice structure. It fact, we shall be interested in the minimal distances which can be deduced from a set of distance inequalities. It is easy to see that the following rules are sound:

$$\frac{}{d(\tau, \tau) \leqslant \langle 1, 1 \rangle} \tag{Refl}$$

$$\frac{d(\tau_1, \tau_2) \leqslant e \in \mathcal{I}}{\mathcal{I} \ \vdash \ d(\tau_1, \tau_2) \leqslant e} \tag{Ax}$$

$$\frac{\mathcal{I} \ \vdash \ d(\tau_1, \tau_2) \leqslant e}{\mathcal{I} \ \vdash \ d(\tau_2, \tau_1) \leqslant e^{-1}} \tag{Dual}$$

$$\frac{\mathcal{I} \ \vdash \ d(\tau_1, \tau_2) \leqslant e_1 \qquad \mathcal{I} \ \vdash \ d(\tau_2, \tau_3) \leqslant e_2}{\mathcal{I} \ \vdash \ d(\tau_1, \tau_3) \leqslant e_1 + e_2} \tag{Join}$$

$$\frac{\mathcal{I} \ \vdash \ d(\tau_1, \tau_2) \leqslant e_1 \qquad \mathcal{I} \ \vdash \ d(\tau_1, \tau_2) \leqslant e_2}{\mathcal{I} \ \vdash \ d(\tau_1, \tau_2) \leqslant e_1 \wedge e_2} \tag{Min}$$

Since we are only interested in minimal distance between distinct elements of Q, use of Refl is unnecessary. If $\mathcal{I} \ \vdash \ d(\tau_1, \tau_2) \leqslant e$ for some distance e, we will write $d_{\mathcal{I}}(\tau_1, \tau_2)$ for the least such distance; in the presence of Min this is well-defined.

2.3.1 Definition [Distance Consistency]: We say that a set of distance inequalities $\mathcal{I}$ is *distance consistent* over Q if for any two elements $\mathtt{a}$ and $\mathtt{b}$ of Q, $d_Q(\mathtt{a}, \mathtt{b}) \leqslant d_{\mathcal{I}}(\mathtt{a}, \mathtt{b})$.

A set of distance inequalities $\mathcal{I}$ is *satisfiable* if there is a ground substitution such that $d_Q(\theta\tau_1, \theta\tau_2) \leqslant e$ for every $d(\tau_1, \tau_2) \leqslant e$ in $\mathcal{I}$. θ is then a *satisfying substitution*, or, analogously with constraint sets, a *solution*.

2.3.2 Proposition: Let $\mathcal{I}$ be a set of inequalities over a Helly poset Q containing inequalities only between variables and elements of Q. Then if $\mathcal{I}$ is distance consistent, $\mathcal{I}$ is satisfiable.

Proof: Let C be a set of constraints (possibly containing fresh variables) which make up the fences for the distances in $\mathcal{I}$, and $\equiv_C$ be defined by

$$\tau_1 \equiv_C \tau_2 \quad \text{if } d_{\mathcal{I}}(\tau_1, \tau_2) = \langle 1, 1 \rangle$$

We can define a poset Q' by quotienting C by this equivalence relation, so that distances in Q' are the same as those in C (and those in $\mathcal{I}$). Then $d_Q(\mathtt{a},\mathtt{b}) \leqslant d_{Q'}(\mathtt{a},\mathtt{b})$, so $Q' \rhd Q$. $\square$

2.3.3 Definition: A *chain* in a set of distance inequalities $\mathcal{I}$ from τ to τ' of length e is a set of distinct elements $\{\tau_1, \ldots \tau_n\}$ such that $\tau = \tau_1$, $\tau' = \tau_n$, and for each i, either $d(\tau_i, \tau_{i+1}) \leqslant e_i \in \mathcal{I}$ or $d(\tau_{i+1}, \tau_i) \leqslant e^{-1} \in \mathcal{I}$. and $\sum_1^{n-1} e_i = e$. The *length* of the chain is e, the *size* of the chain is n. If Q is a Helly poset, and there is a chain of length e in $\mathcal{I}$ between $\mathtt{a}$ and $\mathtt{b}$ such that $d_Q(\mathtt{a},\mathtt{b}) \not\leqslant e$, we say that the chain *witnesses* the inconsistency of $\mathcal{I}$ over Q.

2.3.4 Proposition: $d_{\mathcal{I}}(\tau_1, \tau_2)$ is the minimum of the lengths of all chains in $\mathcal{I}$ from τ_1 to τ_2.

Proof: $d_{\mathcal{I}}(\tau_1, \tau_2)$ is built from the distances in $\mathcal{I}$ using minimisation, addition and reversal. Using the identities

$$\begin{array}{rcl} (e_1 + e_2)^{-1} & = & e_1^{-1} + e_2^{-1} \\ (e_1 \wedge e_2)^{-1} & = & e_1^{-1} \wedge e_2^{-1} \\ e_1 + (e_2 \wedge e_2) & = & (e_1 + e_2) \ \wedge \ (e_1 + e_3) \\ (e_1 + e_2) \wedge e_3 & = & (e_1 + e_3) \ \wedge \ (e_2 + e_3) \end{array}$$

we can permute uses of minimisation outwards over addition and reversal, and addition outwards over reversal, so that the minimal distance can always be expressed as the minimum of the lengths of a set of chains. $\square$

2.3.5 Proposition: Suppose $\mathcal{I}$ is not distance consistent over Q. Then there is some chain $\mathtt{a} = \tau_1, \ldots \tau_n = \mathtt{b}$ witnessing the inconsistency of $\mathcal{I}$ such that $\tau_2, \ldots \tau_{n-1}$ are all variables.

Proof: Let $\{e_i\}$ be the lengths of all the chains from $\mathtt{a}$ to $\mathtt{b}$ in $\mathcal{I}$. If $d_Q(\mathtt{a},\mathtt{b}) \leqslant e_i$ for all i, then $d_Q(\mathtt{a},\mathtt{b}) \leqslant \bigwedge e_i$, so $d_Q(\mathtt{a},\mathtt{b}) \leqslant d_{\mathcal{I}}(\mathtt{a},\mathtt{b})$. Thus if there is no chain witnessing the inconsistency of $\mathcal{I}$ over Q, $\mathcal{I}$ must be consistent over Q.

So suppose $\mathcal{I}$ is not distance consistent over Q, so that there must be some witness chain. Let X be a witness chain of minimal size, and suppose X starts at $\mathtt{a}$, ends at $\mathtt{b}$, and is of length e. If X contains no base types other than $\mathtt{a}$ and $\mathtt{b}$, the proposition holds.

Otherwise X contains some base type $\mathtt{c}$. Let X_1 be the chain from $\mathtt{a}$ to $\mathtt{c}$, and X_2 from $\mathtt{c}$ to $\mathtt{b}$, with lengths e_1 and e_2 respectively. If we have $d_Q(\mathtt{a},\mathtt{c}) \leqslant e_1$ and $d(\mathtt{c},\mathtt{b}) \leqslant e_2$, then $d_Q(\mathtt{a},\mathtt{b}) \leqslant e_1 + e_2 = e$, which cannot be the case since X witnesses the inconsistency of $\mathcal{I}$. Thus either X_1 or X_2 must also be a witness to the inconsistency of $\mathcal{I}$, contradicting the minimality of X. $\square$

Thus if we have a set $\mathcal{I}$ of distance inequalities between variables and elements of Q, it suffices for distance consistency that the $\mathcal{I}$ contains no chain having variables other than at its endpoints which witnesses its inconsistency.

2.4 Solving Constraint Sets

Over a lattice, solving sets of distance inequalities is quite straightforward.

2.4.1 Definition: Let $\mathcal{I}$ be a set of distance inequalities over a lattice Q. We define

$$\mathrm{U}(\alpha) = \bigwedge \{\mathtt{a} \in Q \mid d_{\mathcal{I}}(\alpha, \mathtt{a}) \leqslant \langle 1, 2\rangle\}$$

and

$$\mathrm{D}(\alpha) = \bigvee \{\mathtt{a} \in Q \mid d_{\mathcal{I}}(\alpha, \mathtt{a}) \leqslant \langle 2, 1\rangle\}$$

2.4.2 Lemma: Let Q be a lattice, and $\mathcal{I}$ be a set of distance inequalities between variables and elements of Q which contains Q. If $\mathcal{I}$ is solvable, then $\theta(\alpha) = \mathrm{U}(\alpha)$ is a solution of $\mathcal{I}$.

Proof: Suppose not, so there are e, τ_1 and τ_2 such that $d_{\mathcal{I}}(\tau_1, \tau_2) \leqslant e$ but $d_Q(\theta\tau_1, \theta\tau_2) \nleqslant e$. If $e = \langle 1, 1\rangle$, the result is immediate. If $e = \langle 1, 2\rangle$, then for any $\mathtt{a} \geqslant \tau_2$ we must have $\mathtt{a} \geqslant \tau_1$, and therefore $\mathrm{U}(\tau_2) \geqslant \mathrm{U}(\tau_1)$. The $\langle 2, 1\rangle$ case is similar. And since Q is a lattice, any two elements are separated by a distance of at most $\langle 2, 2\rangle$. □

Dually, $\theta(\alpha) = \mathrm{D}(\alpha)$ is also a solution. Thus we have functions which extract a solution at each variable using only the minimal distances between that variable and the base types in Q. We may obtain a similar result for trees:

2.4.3 Lemma: Let Q be a tree, and $\mathcal{I}$ a set of distance inequalities between variables and elements of Q which contains Q. If $\mathcal{I}$ is solvable over Q, then $\theta(\alpha) = \mathrm{U}(\alpha)$ is a solution of $\mathcal{I}$.

Proof: Suppose not, so that there are τ_1 and τ_2 such that $d_{\mathcal{I}}(\tau_1, \tau_2) \leqslant e$ but $d_Q(\theta\tau_1, \theta\tau_2) \nleqslant e$. We will proceed by cases on e.

$\langle 1, 1\rangle$ $\mathrm{U}(\tau_1) = \mathrm{U}(\tau_2)$ by deductive closure, so we must have $\theta(\tau_2) = \theta(\tau_1)$.

$\langle 1, 2\rangle$ (i.e. $\tau_1 \leqslant \tau_2$) Suppose $d_{\mathcal{I}}(\theta\tau_1, \theta\tau_2) > \langle 1, 2\rangle$. There are two possibilities: either $\theta\tau_1 > \theta\tau_2$, in which case there is $\mathtt{a}$ such that $\theta\tau_1 > \mathtt{a} \geqslant \theta\tau_2$, but then we must have $\mathcal{I} \vdash \tau_2 < \mathtt{a}$ and $\mathcal{I} \nvdash \tau_1 \leqslant \mathtt{a}$, which cannot be the case since

$\mathcal{I} \vdash \tau_1 \leqslant \tau_2$. The other possibility is that $\theta\tau_1$ and $\theta\tau_2$ are incomparable, but then again there is some $\mathtt{a}$ such that $\tau_2 \leqslant \mathtt{a}$ and $\tau_1 \nleqslant \mathtt{a}$, which again contradicts the assumption that $\mathcal{I} \vdash \tau_1 \leqslant \tau_2$.

$\langle 2, 1\rangle$ (i.e. $\tau_2 \leqslant \tau_1$) is identical to the above case

$\langle 2, 2\rangle$ If $d_{\mathcal{I}}(\theta\tau_1, \theta\tau_2) \nleqslant \langle 2, 2\rangle$, we must have $d_{\mathcal{I}}(\theta\tau_1, \theta\tau_2) = \langle 2, 3\rangle$, since this is the maximum distance between any two elements of Q. Since $\theta\tau_1$ and $\theta\tau_2$ are not comparable, there must be $\mathtt{a} \geqslant \tau_1$ and $\mathtt{b} \geqslant \tau_2$ such that $\mathtt{a}$ and $\mathtt{b}$ have no lower bound. But then τ_1 and τ_2 cannot possess a lower bound, contradicting the assumption that $d_{\mathcal{I}}(\tau_1, \tau_2) \leqslant \langle 2, 2\rangle$. □

2.4.4 Definition [Helly Solution Functions]: A *Q-state* S is a function $Q \rightharpoonup \mathcal{L}$. A Q-state is *Q-consistent* if it is either everywhere undefined (this state we shall call $S_\perp$) or it is defined on exactly one component of Q, and for all $\mathtt{a}$ and $\mathtt{b}$ in this component, $d_Q(\mathtt{a}, \mathtt{b}) \leqslant S(\mathtt{a})^{-1} + S(\mathtt{b})$. If S and T are Q-states, We say $d_Q(S, T) \leqslant e$ iff $\mathrm{dom}(S) = \mathrm{dom}(T)$, and for every $\mathtt{a} \in \mathrm{dom}(S)$, $S(\mathtt{a}) \leqslant e + T(\mathtt{a})$ and $T(\mathtt{a}) \leqslant e^{-1} + S(\mathtt{a})$

A function $\mathcal{F}_Q$ from Q-consistent states to Q is a Helly Solution Function (HSF) for Q if

- $d_Q(\mathcal{F}_Q(S), \mathtt{a}) \leqslant S(\mathtt{a})$ for any $\mathtt{a}$ such that $S(\mathtt{a})\downarrow$
- for any S and T such that $d_Q(S, T) \leqslant e$, $d_Q(\mathcal{F}_Q(S), \mathcal{F}_Q(T)) \leqslant e$

It is easy to see that the solutions for lattices and trees are examples of Helly solution functions: for example in the case of lattices we define $S_\alpha(\mathtt{a}) = d_{\mathcal{I}}(\alpha, \mathtt{a})$, and

$$\mathcal{F}_Q(S_\alpha) = U(\alpha) = \bigwedge\{\mathtt{a} \mid S_\alpha(\mathtt{a}) \leqslant \langle 1, 2\rangle\}$$

2.4.5 Proposition: Suppose $\mathcal{I}$ is solvable over Q and contains Q, and $\mathcal{F}_Q$ is a HSF for Q. If we define $S_\alpha(\mathtt{a}) = d_{\mathcal{I}}(\alpha, \mathtt{a})$, then $\theta(\alpha) = \mathcal{F}_Q(S_\alpha)$ is a solution of $\mathcal{I}$.

Proof: Since $\mathcal{I}$ contains Q and is distance consistent over Q, each S_α is either $S_\perp$ or defined on exactly one component of Q: moreover since $\mathcal{I}$ is distance consistent over Q, it cannot be the case for any $\mathtt{a}$ and $\mathtt{b}$ that $d_Q(\mathtt{a}, \mathtt{b}) \nleqslant d_{\mathcal{I}}(\mathtt{a}, \alpha) + d_{\mathcal{I}}(\alpha, \mathtt{b})$, so $d_Q(\mathtt{a}, \mathtt{b}) \leqslant S_\alpha(\mathtt{a})^{-1} + S_\alpha(\mathtt{b})$. Thus S_α is a Q-consistent Q-state. Suppose there is some inequality $d(\alpha, \tau) \leqslant e$ in $\mathcal{I}$. There are two possible cases:

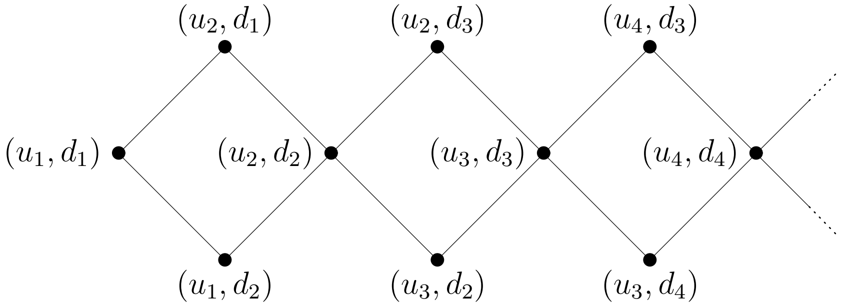
- if τ is a base type $\mathtt{a}$, then $S(\mathtt{a}) \leqslant e$ and so it is immediate from the definition of HSF that $d(\theta\alpha, \mathtt{a}) \leqslant e$.

- if τ is a variable β, by JOIN we have $S_\alpha(\mathtt{a}) \leqslant e + S_\beta(\mathtt{a})$ for any $\mathtt{a} \in \mathrm{dom}(S_\alpha)$, and by DUAL and JOIN, $S_\beta(\mathtt{a}) \leqslant e^{-1} + S_\alpha(\mathtt{a})$. So $d(S_\alpha, S_\beta) \leqslant e$, and again from the definition of HSF, $d(\theta(\alpha), \theta(\beta)) \leqslant e$ □

In fact, we can show that all Helly posets have HSFs. In order to do so, we need a few preliminary results.

2.4.6 Lemma: Suppose U is an up-fence of length m, and D a down-fence of length m. We define

$$T = \{(u,d) \in U \times D \mid |u - d| \leqslant 1\}$$



Then $U \times D \rhd T$.

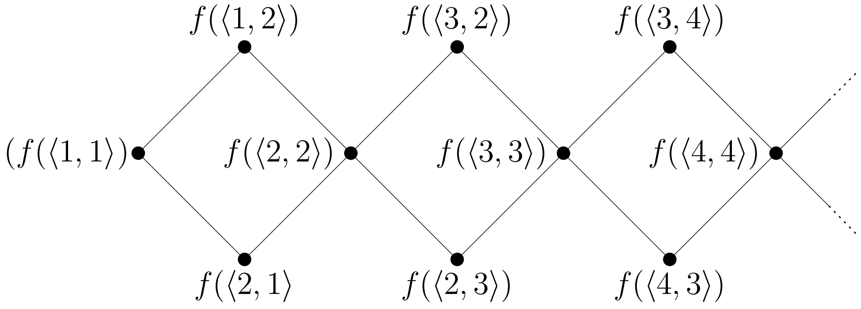
Proof: It is straightforward to check that the map

$$\begin{array}{lcll} (u_i, d_j) & \mapsto & (u_i, d_{i-1}) & \text{if } j \leqslant i-1 \\ (u_i, d_j) & \mapsto & (u_i, d_{i+1}) & \text{if } j \geqslant i+1 \\ (u_i, d_j) & \mapsto & (u_i, d_j) & \text{otherwise} \end{array}$$

is a retraction. □

2.4.7 Lemma: If we define $f : \mathcal{L} \to U \times D$ by

$$f(\langle p, q\rangle) = (u_p, d_q)$$



Suppose $e_1, e_2, e_3 \leqslant \langle m, m\rangle$, then

(a) if $e_1 = \langle n, n\rangle$ for some $n \leqslant m$, then $d(f(e_1), f(e_1+e_2)) = e_2$

(b) if $e_1 \leqslant e_2 \leqslant e_3$, then $d_{U\times D}(f(e_1), f(e_2)) \leqslant d_{U\times D}(f(e_1), f(e_3))$

(c) $d_{U\times D}(f(e_1), f(e_1+e_2)) \leqslant e_2$

(d) If $e_1 + e_2 \geqslant e_3$ and $e_3 + e_2^{-1} \geqslant e_1$, then $d_{U\times D}(f(e_1), f(e_3)) \leqslant e_2$

Proof: By lemma(2.4.6) $U \times D$ is a T-model, so $d_{U\times D}(\mathtt{a}, \mathtt{b}) = d_T(\mathtt{a}, \mathtt{b})$ by proposition(2.2.6). Thus it suffices to demonstrate to result for distances in T.

(a) Let $e_1 = \langle n, n\rangle$, $e_2 = \langle u_2, d_2\rangle$. We proceed by induction on $\max(u_2, d_2)$. If $\max(u_2, d_2) \leqslant 2$, the results for the cases n odd and n even can easily be observed from the diagram. Otherwise suppose the result holds for all smaller values of $\max(u_2, d_2)$ Since $\max(u_2, d_2) > 2$, every fence from $f(e_1)$ to $f(e_1 + e_2)$ must pass through $f(\langle n+1, n+1\rangle)$, and $d_T(e_1, e_1+e_2)$ is then

$$\begin{aligned}
& d_T(f(e_1), f(\langle n+1, n+1\rangle)) + d_T(f(\langle n+1, n+1\rangle), f(e_1+e_2)) \\
= \ & \langle 2,2\rangle + d_T(f(e_1 + \langle 2,2\rangle), f(e_1 + \langle 2,2\rangle + \langle d_2 - 1, u_2 - 1\rangle)) \\
& \quad \text{since } \langle u, d\rangle = \langle 2,2\rangle + \langle d_2 - 1, u_2 - 1\rangle \\
= \ & \langle 2,2\rangle + \langle d_2 - 1, u_2 - 1\rangle \\
& \quad \text{by the induction hypothesis} \\
= \ & \langle u_2, d_2\rangle
\end{aligned}$$

(b) Suppose $e_1 \leqslant e_2 \leqslant e_3$. Let $\mathtt{a}_1 = f(e_1)$, $\mathtt{a}_2 = f(e_2)$, $\mathtt{a}_3 = f(e_3)$. We will show first that the shortest up-fence F from $\mathtt{a}_1$ to $\mathtt{a}_3$ is no longer than the shortest up-fence from $\mathtt{a}_1$ to $\mathtt{a}_2$. Since $\mathtt{a}_3$ is strictly to the right of $\mathtt{a}_2$ in our picture of T, F must pass through either $f(e_2)$ or $\mathtt{a}_2' = f(e_2^*)$. If F passes through $\mathtt{a}_2$, the result is immediate. So suppose not, then $\mathtt{a}_2 \neq \mathtt{a}_2'$, so $e_2 \neq e_2^*$, and moreover we must have $e_3 \geqslant e_2 \vee e_2^*$, so F must pass through $\mathtt{b} = f(e_2 \vee e_2^*)$ But a fence from $\mathtt{a}_1$ to $\mathtt{a}_3$ via $\mathtt{a}_2'$ and $\mathtt{b}$ is at least as long as the fence from $\mathtt{a}_1$ to $\mathtt{a}_2$ via $\mathtt{a}_2'$ and $\mathtt{b}$.

The same reasoning holds for the shortest down-fence from $\mathtt{a}_1$ to $\mathtt{a}_3$, so combining these, we must have $d_T(\mathtt{a}_1, \mathtt{a}_2) \leqslant d_T(\mathtt{a}_1, \mathtt{a}_3)$

(c) if $e_1 = \langle n, n\rangle$, then the result is immediate from (a). Otherwise, let $e_1 = \langle u_1, d_1\rangle$. Suppose u_1 is even and d_1 odd, so that $f(e_1)$ is on the "bottom" of T. Then from the definition of addition we have

$$\begin{aligned}
\langle u_1, d_1\rangle + \langle u_2, d_2\rangle &= \langle u_1 + d_2 - 1, d_1 + d_2 - 1\rangle \\
&= \langle u_1, d_1\rangle + \langle u_2, d_2\rangle^{\vartriangle}
\end{aligned}$$

Thus if $e_2 = \langle 1, 1\rangle$ or $\langle 2, 1\rangle$, $e_1 + e_2 = e_1$ and the result is trivial. So suppose $d_2 > 1$, and let $e = \langle \max(u_1, d_1), \max(u_1, d_1)\rangle$. Observe from the diagram that $d_T(f(e_1), f(e)) = \langle 1, 2\rangle$. Now for any $e' \geqslant \langle 2, 1\rangle$, every fence from $f(e_1)$ to $f(e_1 + e')$ must pass through $f(e)$, so

$$\begin{array}{rcl} d_T(f(e_1), f(e_1 + e_2)) & = & d_T(f(e_1), f(e_1 + e_2^{\vartriangle})) \\ & \leqslant & d_T(f(e_1), f(e)) + d_T(f(e), f(e + e_2^{\vartriangle})) \\ & \leqslant & \langle 1, 2\rangle + e_2^{\vartriangle} \qquad \text{(by part (a))} \\ & = & e_2^{\vartriangle} \end{array}$$

The final equality holds because since $d_2 > 1$, $e_2^{\vartriangle} = \langle d_2 - 1, d_2\rangle$, and $\langle 1, 2\rangle + \langle d_2 - 1, d_2\rangle = \langle d_2 - 1, d_2\rangle$. The result follows dually if u_1 is odd and d_1 even.

(d) Suppose $e_1 + e_2 \geqslant e_3$, $e_3 + e_2^{-1} \geqslant e_1$. If $e_3 \geqslant e_1$, then since $e_1 + e_2 \geqslant e_3$ we have $d(f(e_1), f(e_3)) \leqslant d(f(e_1), f(e_1 + e_2))$ (by (b)) and thus $d(f(e_1), f(e_3)) \leqslant e_2$ (by (c) and transitivity). Similarly if $e_1 \geqslant e_3$, then since $e_3 + e_2^{-1} \geqslant e_1$, we have we have $d(f(e_3), f(e_1)) \leqslant d(f(e_3), f(e_3 + e_2^{-1})) \leqslant e_2^{-1}$, and thus $d(f(e_1), f(e_3)) \leqslant e_2$.

Finally if e_1 and e_3 are incomparable, $e_1 = e_3^*$. If $e_1 = \langle u, d\rangle$ with u even and d odd, then observe from the diagram that we must have $e_2 \geqslant \langle 1, 2\rangle$, and the result follows immediately. The case where u is odd and d even is dual. □

2.4.8 Proposition: Suppose $\mathcal{I}$ includes Q, and is solvable over Q. Then there is a Helly poset Q' such that $Q' \rhd Q$, and a solution θ of $\mathcal{I}$ over Q' such that the inequalities in $\mathcal{I}$ are satisfied exactly.

Proof: We will construct Q' one component at a time. Suppose $\mathcal{I}$ is connected. Let $\langle m, m\rangle$ be longer than any distance in $\mathcal{I}$, U and D be as in lemma(2.4.6), and the types occurring in $\mathcal{I}$ be $\tau_1 \ldots \tau_k$. We define $Q' = (U \times D)^k$, with

$$\theta(\tau_i) = (f(d_{\mathcal{I}}(\tau_1, \tau_i)), f(d_{\mathcal{I}}(\tau_2, \tau_i)), \ldots, f(d_{\mathcal{I}}(\tau_k, \tau_i))$$

so that

$$d_{Q'}(\theta\tau_1, \theta\tau_2) = \bigvee_{i \leqslant k} (d_{U\times D}(f(d(\tau_i, \tau_1)), f(d(\tau_i, \tau_2)))$$

θ is obviously injective, so it suffices to show that it is distance preserving (for then it is obviously order-preserving). So suppose $d_{\mathcal{I}}(\tau_i, \tau_j) = e$. Then $\theta(\tau_i)$ has $f(\langle 1, 1\rangle)$ in the ith position, and $\theta(\tau_j)$ has $f(e)$, so $d_{Q'}(\theta(\tau_i), \theta(\tau_j))$ is at least e by lemma(2.4.7(a))

On the other hand, since $\mathcal{I}$ is closed under Join and Min, for any τ_h we have $d_{\mathcal{I}}(\tau_h, \tau_i) + e \geqslant d_{\mathcal{I}}(\tau_h, \tau_j)$ and $d_{\mathcal{I}}(\tau_h, \tau_j) + e^{-1} \geqslant d_{\mathcal{I}}(\tau_h, \tau_i)$, thus by lemma(2.4.7(d))

$$d_{U\times D}(f(d_{\mathcal{I}}(\tau_h, \tau_i)), f(d_{\mathcal{I}}(\tau_h, \tau_j))) \leqslant e$$

So in each position h in the k-tuples $\theta(\tau_1)$ and $\theta(\tau_j)$, we have some $e' \leqslant e$. Thus we obtain exactly $d_{Q'}(\theta(\tau_i), \theta(\tau_j)) = e$. Finally, since $\mathcal{I}$ contains Q, and the distances between elements of Q in Q' are the same as those in Q, θ is an embedding of Q in Q' and Q' is distance consistent over Q, so we have $Q' \rhd Q$.

If $\mathcal{I}$ is not connected, we define Q' to be the direct sum of the products for each component. The distance inequalities in $\mathcal{I}$ are met exactly, and $Q' \rhd Q$. □

2.4.9 Definition: A variety V is *generated* by a subset $U \subseteq V$ if every element of V can be constructed from U using products and retractions.

As a simple corollary of proposition(2.4.8), we have the following, which is also proven in [30]

2.4.10 Proposition: The variety of finite connected Helly posets is generated by the set of fences.

Proof: Just take $\mathcal{I}$ in proposition(2.4.8) to be the closure of Q under Dual, Min, and Join. Then Q' is a product of fences such that $Q' \rhd Q$, and since every fence is Helly and Helly posets are closed under products, the result is immediate. □

We can now show that all Helly posets have HSFs

2.4.11 Proposition: If Q is a fence, Q has a HSF.

Proof: Suppose Q is an up-fence $\mathtt{a}_1 \leqslant \mathtt{a}_2 \geqslant \mathtt{a}_3 \ldots$. If S is a Q-consistent Q-state, the set of inequalities

$$\{d(\alpha, \mathtt{a}) \leqslant S(\mathtt{a}) \mid S(\mathtt{a})\!\downarrow\}$$

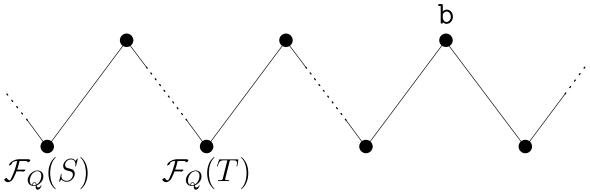
is distance consistent, and thus solvable over Q. So we define

$$j(S) = \min\{i \mid \forall \mathtt{b} \in Q.\, d_Q(\mathtt{a}_i, \mathtt{b}) \leqslant S(\mathtt{b})\}$$

and set $\mathcal{F}_Q(S) = \mathtt{a}_{j(S)}$.

Now suppose $\mathcal{F}_Q$ is not a HSF for $\mathcal{Q}$, so that we have Q-states S and T and $e \in \mathcal{L}$ such that $d_Q(S, T) \leqslant e$ but $d_Q(\mathcal{F}_Q(S), \mathcal{F}_Q(T) \nleqslant e$. Obviously we cannot have $S = T = S_\perp$, and we cannot have $\mathcal{F}_Q(S) = \mathcal{F}_Q(T)$, so $j(S) \neq (T)$. We may assume $j(S) < j(T)$.

Let $\mathtt{b}$ be such that $d_Q(\mathtt{a}_{j(T)}, \mathtt{b}) \leqslant T(\mathtt{b})$, but $d_Q(\mathtt{a}_{j(T)-1}, \mathtt{b}) \nleqslant T(\mathtt{b})$, in other words $\mathtt{b}$ is an obstacle to $j(T)$ being any smaller. Suppose that $j(S)$ and $j(T)$ are both odd, so that $d(\mathcal{F}_Q(S), \mathcal{F}_Q(T)) = \langle 2n, 2n+1 \rangle$, so $d(\mathcal{F}_Q(T), \mathtt{b}) = \langle m, m+1 \rangle$, and $d_Q(\mathcal{F}_Q(S), \mathtt{b}) = \langle 2n+m, 2n+m+1 \rangle$.



Since $\mathtt{b}$ is an obstacle to $j(T)$ being any smaller, $\langle m+2, m+1\rangle \not\leqslant T(\mathtt{b})$, and so $T(\mathtt{b}) \leqslant \langle m+1, m+2\rangle$. And since $d_Q(\mathcal{F}_Q(S), \mathcal{F}_Q(T)) \not\leqslant e$, $\langle 2n, 2n+1\rangle \not\leqslant e$, so $e \leqslant \langle 2n+1, 2n\rangle$. Thus by monotonicity of addition, $S(\mathtt{b}) \leqslant \langle 2n+1, 2n\rangle + \langle m+1, m+2\rangle = \langle 2n+m+1, 2n+m\rangle$.

But by the choice of $\mathcal{F}_Q(S)$, $d_Q(\mathcal{F}_Q(S), \mathtt{b}) = \langle 2n+m, 2n+m+1\rangle \leqslant S(\mathtt{b})$, a contradiction. The cases where $j(S)$ and $j(T)$ are not both odd result in similar contradictions. □

2.4.12 Proposition: If Q_1 and Q_2 are connected and have HSFs, $Q = Q_1 \times Q_2$ has a HSF.

Proof: We define

$$\begin{array}{rcl}\pi_1(S)(\mathtt{a}_1) & = & \bigwedge_{\mathtt{a}_2\in Q_2}\{S(\mathtt{a}_1,\mathtt{a}_2))\} \\ \pi_2(S)(\mathtt{a}_2) & = & \bigwedge_{\mathtt{a}_1\in Q_1}\{S(\mathtt{a}_1,\mathtt{a}_2))\}\end{array}$$

and we claim that

$$\mathcal{F}_Q(S) = (\mathcal{F}_{Q_1}(\pi_1(S)), \mathcal{F}_{Q_2}(\pi_2(S)))$$

is a HSF for Q. Let S and T be Q-states. If $S = T = S_\perp$, then the result is trivial. Otherwise, let $S_1 = \pi_1(S)$, $T_1 = \pi_1(T)$. Since $\mathrm{dom}(S) = \mathrm{dom}(T) = Q$ $\mathrm{dom}(S_1)$ and $\mathrm{dom}(T_1) = Q_1$, and for any $\mathtt{a} = (\mathtt{a}_1, \mathtt{a}_2), \mathtt{b} = (\mathtt{b}_1, \mathtt{b}_2) \in Q$, we have $d_Q(\mathtt{a}, \mathtt{b}) = d_{Q_1}(\mathtt{a}_1, \mathtt{b}_1) \vee d_{Q_2}(\mathtt{a}_2, \mathtt{b}_2)$, so for any $\mathtt{a}_2$ and $\mathtt{b}_2$ we have

$$d_{Q_1}(\mathtt{a}_1, \mathtt{b}_1) \leqslant S((\mathtt{a}_1, \mathtt{a}_2))^{-1} + S((\mathtt{b}_1, \mathtt{b}_2))$$

so

$$d_{Q_1}(\mathtt{a}_1, \mathtt{b}_1) \leqslant \bigwedge_{\mathtt{a}_2\in Q_2} S((\mathtt{a}_1, \mathtt{a}_2))^{-1} + \bigwedge_{\mathtt{b}_2\in Q_2} S((\mathtt{b}_1, \mathtt{b}_2))$$

thus

$$d_{Q_1}(\mathtt{a}_1, \mathtt{b}_1) \leqslant \Big(\bigwedge_{\mathtt{a}_2\in Q_2} S(\mathtt{a}_1, \mathtt{a}_2)\Big)^{-1} + \bigwedge_{\mathtt{b}_2\in Q_2} S((\mathtt{b}_1, \mathtt{b}_2))$$

so $d_{Q_1}(\mathtt{a}_1, \mathtt{b}_1) \leqslant S_1(\mathtt{a}_1)^{-1} + S_1(\mathtt{b}_1)$, and thus S_1 is Q_1-consistent; similarly for T_1. Moreover, if we have e such that $S((\mathtt{a}_1, \mathtt{a}_2)) \leqslant e + T((\mathtt{a}_1, \mathtt{a}_2))$ and $T((\mathtt{a}_1, \mathtt{a}_2)) \leqslant e^{-1} + S((\mathtt{a}_1, \mathtt{a}_2))$, then for any $\mathtt{a}_1 \in Q_1$ we have

$$\begin{array}{rcl}S_1(\mathtt{a}_1) & = & \bigwedge_{\mathtt{a}_2\in Q_2}\{S(\mathtt{a}_1,\mathtt{a}_2))\} \\ & \leqslant & \bigwedge_{\mathtt{a}_2\in Q_2}\{e + T(\mathtt{a}_1,\mathtt{a}_2))\} \\ & \leqslant & e + \bigwedge_{\mathtt{a}_2\in Q_2}\{T(\mathtt{a}_1,\mathtt{a}_2))\} \\ & \leqslant & e + T_1(\mathtt{a}_1)\end{array}$$

and similarly $T_1(\mathtt{a}_1) \leqslant e^{-1} + S_1(\mathtt{a}_1)$. So $d_{Q_1}(\mathcal{F}_{Q_1}(S_1), \mathcal{F}_{Q_1}(T_1)) \leqslant e$. By a similar argument it is straightforward that $d_{Q_2}(\mathcal{F}_{Q_2}(\pi_2(S)), \mathcal{F}_{Q_2}(\pi_2(T))) \leqslant e$, and thus $d_Q(\mathcal{F}_Q(S), \mathcal{F}_Q(T)) \leqslant e$. □

2.4.13 Proposition: If Q_1 and Q_2 are connected and Helly, $r : Q_1 \rhd Q_2$, and Q_1 has a HSF, Q_2 has a HSF

Proof: For S a Q_2-state, we define

$$i(S)(\mathtt{a}) = \bigwedge_{\mathtt{b} \in Q_2} S(\mathtt{b}) + d_{Q_1}(\mathtt{b}, \mathtt{a})$$

and

$$\mathcal{F}_{Q_2}(S) = r(\mathcal{F}_{Q_1}(i(S)))$$

and will show that $\mathcal{F}_{Q_2}$ is a HSF for Q_2. Let S and T be Q_2-consistent Q_2-states such that $\mathrm{dom}(S) = \mathrm{dom}(T)$. If $S = T = S_\perp$, the result is immediate. Otherwise it is straightforward that $\mathrm{dom}(i(S)) = \mathrm{dom}(i(T)) = Q_1$. Now

$$\begin{array}{rcl} i(S)(\mathtt{a})^{-1} + i(S)(\mathtt{b}) & = & \left(\bigwedge_{\mathtt{c} \in Q_2} S(\mathtt{c}) + d_{Q_1}(\mathtt{c}, \mathtt{a})\right)^{-1} + \bigwedge_{\mathtt{d} \in Q_2} S(\mathtt{d}) + d_{Q_1}(\mathtt{d}, \mathtt{b}) \\ & = & \bigwedge_{\mathtt{c},\mathtt{d} \in Q_2} d_{Q_1}(\mathtt{a}, \mathtt{c})) + S(\mathtt{c})^{-1} + S(\mathtt{d}) + d_{Q_1}(\mathtt{d}, \mathtt{b})) \end{array}$$

But since $Q_1 \rhd Q_2$,

$$\begin{array}{rcl} d_{Q_1}(\mathtt{a}, \mathtt{b}) & \leqslant & \bigwedge_{\mathtt{c},\mathtt{d} \in Q_2} d_{Q_1}(\mathtt{a}, \mathtt{c}) + d_{Q_1}(\mathtt{c}, \mathtt{d}) + d_{Q_1}(\mathtt{d}, \mathtt{b}) \\ & = & \bigwedge_{\mathtt{c},\mathtt{d} \in Q_2} d_{Q_1}(\mathtt{a}, \mathtt{c}) + d_{Q_2}(\mathtt{c}, \mathtt{d}) + d_{Q_1}(\mathtt{d}, \mathtt{b}) \\ & \leqslant & \bigwedge_{\mathtt{c},\mathtt{d} \in Q_2} d_{Q_1}(\mathtt{a}, \mathtt{c}) + S(\mathtt{c})^{-1} + S(\mathtt{d}) + d_{Q_1}(\mathtt{d}, \mathtt{b}) \end{array}$$

So $i(S)$ and similarly $i(T)$ are Q_1-consistent Q_1-states. Now if we have e such that $S(\mathtt{a}) \leqslant e + T(\mathtt{a})$ and $T(\mathtt{a}) \leqslant e^{-1} + S(\mathtt{a})$ for all $\mathtt{a} \in Q_2$, then for all $\mathtt{a} \in Q_1$, we have

$$\begin{array}{rcl} i(S)(\mathtt{a}) & = & \bigwedge_{\mathtt{b} \in Q_2} S(\mathtt{b}) + d_{Q_1}(\mathtt{b}, \mathtt{a}) \\ & \leqslant & \bigwedge_{\mathtt{b} \in Q_2} e + T(\mathtt{b}) + d_{Q_1}(\mathtt{b}, \mathtt{a}) \\ & \leqslant & e + (\bigwedge_{\mathtt{b} \in Q_2} T(\mathtt{b}) + d_{Q_1}(\mathtt{b}, \mathtt{a})) \\ & \leqslant & e + i(T)(\mathtt{a}) \end{array}$$

and similarly $i(T)(\mathtt{a}) \leqslant e^{-1} + i(S)(\mathtt{a})$. Thus $d_{Q_1}(\mathcal{F}_{Q_1}(S), \mathcal{F}_{Q_1}(T)) \leqslant e$, and since $Q_1 \rhd Q_2$, by proposition(2.2.5), $d_{Q_2}(r(\mathcal{F}_{Q_1}(S)), r(\mathcal{F}_{Q_1}(T))) \leqslant e$, i.e. $d_{Q_2}(\mathcal{F}_{Q_2}(S), \mathcal{F}_{Q_2}(T)) \leqslant e$. □

2.4.14 Proposition: Every Helly poset Q has a HSF.

Proof: By proposition(2.4.10), using propositions (2.4.11), (2.4.12), and (2.4.13), each connected component Q_i of Q has a HSF. We choose some element $\mathtt{v} \in Q$ and define $\mathcal{F}_Q(S_\perp) = \mathtt{v}$. For any other Q-state S, $\mathrm{dom}(S) = Q_i$ for some i, and since S is Q-consistent, it is Q_i-consistent, and we may apply the HSF for Q_i. □

In chapter 4 we will extend this result to consider solvability and solution for constraint sets over types built with type constructors.

Chapter 3

Simplification and Entailment

In this chapter we will deal with the question of what it means for a constraint set C_1 to entail a constraint set C_2, and how the relation can be tested. We will only work with atomic constraint sets: if the subtyping system involves only structural constructors over a poset of atomic types, it is straightforward using substitution (although potentially expensive) to reduce any $\mathrm{ML}_{\leqslant}$ typing judgement to atomic form. It is easy to demonstrate that the reduction yields a halbstark instance of the original typing judgement, and that all ground instances of the original judgement are instances of its atomic reduction.

There are several possible ways in which we might characterise entailment: however a property we would certainly intuitively expect can be stated informally as follows: every valid interpretation (i.e. solution) of C_1 is a solution of C_2.

Consider the term $\lambda f \lambda x.\, f(fx)$. Our type inference algorithm yields the type $\alpha{\to}\beta{\to}\epsilon$ with the constraint set $C = \{\alpha{\leqslant}\beta{\to}\gamma, \alpha{\leqslant}\delta{\to}\epsilon, \gamma{\leqslant}\delta\}$. By the substitution $[\alpha_1{\to}\alpha_2/\alpha]$ we can reduce this to atomic form:

$$C' = \{\beta{\leqslant}\alpha_1, \alpha_2{\leqslant}\gamma, \delta{\leqslant}\alpha_1, \alpha_2{\leqslant}\epsilon, \gamma{\leqslant}\delta\}$$

By principality, the types at which we can use this expression are precisely those for which the constraints are solvable. But in such solutions, we only care about the values of α, β, and ϵ (or in the atomic form of the judgement, α_1, α_2, γ and ϵ). Given particular values for these variables, as long as there are *some* values for γ and δ which satisfy the constraints, we don't really care what they are. We can think of the variables α, β, and ϵ as externally observable, in the sense that in any solution their exact values are reflected in the type of the expression, whereas the values of γ and δ are internally hidden.

Each constraint set with which we deal will have some set of variables which we consider *internal*; the rest will be *external*. In a judgement $\Gamma; C_1 \vdash e : \forall\overline{\alpha}\backslash C_2.\tau$, the internal variables will be those occurring in $\overline{\alpha}$ but not in τ; the externals will be the others: in our type inference algorithm they will be those occurring in τ and Γ, since these are the variables which can be "observed" by the operation of the inference algorithm. We will refer to the set of base types and external variables together as the *observables*.

A further consideration for entailment is the class of posets over which instantiation can occur. The requirement that all solutions of C_1 have solutions of C_2 which are equal on externals over a class $\mathcal{Q}$ of posets becomes stronger as $\mathcal{Q}$ becomes larger. For example, if we know we will only work over discrete posets, then all constraints will be instantiated as equalities and the entailment relation will be relatively weak (i.e. it relates a lot of constraint sets); on the other hand if entailment over all posets is required, then the relation is much stronger (i.e. relates fewer constraint sets). The merit of a weak entailment relation is that we obtain more equivalences (equivalence just being entailment in both directions) and thus more effective ways of finding small representations of constraint sets, i.e. we get better simplification. For example we have seen that simplifications can be made to the type of `max` if we only consider instantiations over a Helly poset; if we must also consider the 2-crown, no simplification is possible. And the constraint sets $\{\alpha \leqslant \mathtt{a}\}$ and $\{\alpha \leqslant \mathtt{a}, \mathtt{a} \leqslant \alpha\}$ have the same solutions for α over a tree Q in which if `a` is a leaf. But if we need to consider possible future extensions of the tree where there is some type $\mathtt{b} < \mathtt{a}$, the two constraint sets are clearly not equivalent: $[\mathtt{b}/\alpha]$ is a solution of one but not the other.

We will work with Helly posets, and since we are interested in languages with an extensible subtyping relation, we will consider entailment relations which are valid over various classes of Q-models for some Helly poset Q which represents the current ordering. We will thus always assume that Q is included in the constraint set, since by proposition(2.2.6) this doesn't change the set of solutions over any Q-model.

We will start with a study of entailment and equivalence of constraint sets, and then proceed to define our simplification technique.

3.1 Entailment

Suppose we have two constraint sets C_1 and C_2 which share the same set $\overline{\alpha}$ of externals. We may regard the internal variables as implicitly existentially quantified: If C is a constraint set and Q is a poset, we will say $Q \models C$ if there is a ground substitution $\theta : \mathrm{fv}(C) \rightarrow Q$ such that all the constraints in θC are satisfied over Q. Then if $\overline{\alpha}$ is the set of externals in C, we will say $\theta{:}\overline{\alpha}{\rightarrow}Q$ is a

$(Q, \overline{\alpha})$-*solution* of C for $\overline{\alpha}$ over Q if $Q \models \theta C$.

Our relation will be parameterised by a set of externals $\overline{\alpha}$ and a class of posets $\mathcal{Q}$. Given constraint sets C_1 and C_2, we say C_1 *entails* C_2 for $\overline{\alpha}$ over $\mathcal{Q}$ and write $C_1 \models^{\mathcal{Q}}_{\overline{\alpha}} C_2$ iff

$$\forall Q \in \mathcal{Q}\, \forall \theta{:}\overline{\alpha}{\to}Q \;\; (Q \models \theta C_1 \Rightarrow Q \models \theta C_2)$$

Then C_1 and C_2 are equivalent (for $\overline{\alpha}$ over $\mathcal{Q}$) if $C_1 \models^{\mathcal{Q}}_{\overline{\alpha}} C_2$ and $C_2 \models^{\mathcal{Q}}_{\overline{\alpha}} C_1$.

As might be expected, since we are considering Helly posets, we can characterise the set of $(Q, \overline{\alpha})$-solutions of a constraint set C in terms of the minimal distances between the observables. In fact, we have a stronger result: that we can also characterise the $(Q', \overline{\alpha})$-solutions of C over any $Q' \rhd Q$ in terms of these distances.

3.1.1 Proposition: Suppose $\mathcal{I}$ is a set of distance inequalities which is distance consistent over Q, Q' is a Q-model, and $\overline{\alpha}$ is the set of externals in $\mathcal{I}$. Then for any substitution $\theta : \overline{\alpha} \to Q'$, $\theta\mathcal{I}$ is distance consistent (and hence solvable) iff for every pair $\tau_1, \tau_2 \in Q \cup \overline{\alpha}$, $d_{Q'}(\theta\tau_1, \theta\tau_2) \leqslant d_{\mathcal{I}}(\tau_1, \tau_2)$.

Proof: If $\theta\mathcal{I}$ is distance consistent over Q', then for any pair $\tau_1, \tau_2 \in Q \cup \overline{\alpha}$, $d_{Q'}(\theta\tau_1, \theta\tau_2) \leqslant d_{\theta\mathcal{I}}(\theta\tau_1, \theta\tau_2)$. Any proof $\mathcal{I} \vdash d(\tau_1, \tau_2) \leqslant e$ translates to a proof of $\theta\mathcal{I} \vdash d(\theta\tau_1, \theta\tau_2) \leqslant e$, so $d_{\theta\mathcal{I}}(\theta\tau_1, \theta\tau_2) \leqslant d_{\mathcal{I}}(\tau_1, \tau_2)$, and thus $d_{Q'}(\theta\tau_1, \theta\tau_2) \leqslant d_{\mathcal{I}}(\tau_1, \tau_2)$.

Conversely, suppose for every pair $\tau_1, \tau_2 \in Q \cup \overline{\alpha}$, $d_{Q'}(\theta\tau_1, \theta\tau_2) \leqslant d_{\mathcal{I}}(\tau_1, \tau_2)$. If $\theta\mathcal{I}$ is not solvable, then by proposition(2.3.5) there must be some chain in $\theta\mathcal{I}$ between $\theta\tau_1$ and $\theta\tau_2$ of length e containing only variables other than at its endpoints, such that $d_{Q'}(\theta\tau_1, \theta\tau_2) \nleqslant e$. But these variables are exactly the internals in $\mathcal{I}$, so the chain exists in $\mathcal{I}$, and thus $d_{\mathcal{I}}(\tau_1, \tau_2) \leqslant e$, a contradiction to the assumption $d_{Q'}(\theta\tau_1, \theta\tau_2) \leqslant d_{\mathcal{I}}(\tau_1, \tau_2)$. □

Tautness

In order to characterise the solutions of constraint set, we will use a notion of closure:

3.1.2 Definition [tautness]: A set $\mathcal{I}$ of distance inequalities is *taut* for some set of variables $\overline{\alpha}$ over some class $\mathcal{Q}$ of posets if for any inequality $d(\tau_1, \tau_2) \leqslant e$ in $\mathcal{I}$ and any $e' < e$, there is a poset $Q \in \mathcal{Q}$ and a $(Q, \overline{\alpha})$-solution of $\mathcal{I}$ which is not a $(Q, \overline{\alpha})$-solution to $\mathcal{I} \cup \{d(\tau_1, \tau_2) \leqslant e'\}$. $\mathcal{J}$ is a *tautening* of $\mathcal{I}$ on $\overline{\alpha}$ over $\mathcal{Q}$ if it is taut, and equivalent to $\mathcal{I}$ for $\overline{\alpha}$ over $\mathcal{Q}$.

If $\mathcal{I}$ and $\mathcal{J}$ are both taut for some set of variables $\overline{\alpha}$ over $\mathcal{Q}$, then there is no scope for decreasing any of the distances between observables without losing some solutions. So they are equivalent over $\mathcal{Q}$ iff the minimum distances between their observables are the same. Moreover, the set of solutions of $\mathcal{I}$ will be a strict subset of the solution set of $\mathcal{J}$ if some of the distances in $\mathcal{I}$ are shorter than those of $\mathcal{J}$ and none are longer. Thus the notions of entailment and equivalence can be tested by computing tautenings.

Over a poset Q, the weakest (in the sense of giving fewest entailment judgements) equivalence that we shall study using this notion is equivalence over the whole class of Helly posets which retract to Q. We have the following easy result:

3.1.3 Proposition: Suppose $\mathcal{I}$ is a set of distance inequalities containing Q and solvable over Q, and closed under Dual, Min, and Join, and $\overline{\alpha}$ is some set of external variables in $\mathcal{I}$. Then $\mathcal{I}$ is taut for $\overline{\alpha}$ over the class of Helly posets which retract to Q.

Proof: By proposition(2.4.8) there is a Helly poset $Q' \rhd Q$ in which the distances in $\mathcal{I}$ between the observables $\overline{\alpha} \cup Q$ are satisfied exactly, and by decreasing any of these distances this solution is lost. $\square$

3.2 Simplification

Suppose we are interested in simplifications which preserve all solutions over a particular class of posets. Returning to the example of `max`, defining $C = \{\alpha \leqslant \texttt{Real}, \beta \leqslant \texttt{Real}, \alpha \leqslant \gamma, \beta \leqslant \gamma\}$, we have

$$\varnothing; C \vdash \texttt{max} : \alpha \to \beta \to \gamma \quad \ll \quad \varnothing; \delta \leqslant \texttt{Real} \vdash \texttt{max} : \delta \to \delta \to \delta$$

with the substitution $\theta(\alpha) = \theta(\beta) = \theta(\gamma) = \delta$. But it is easy to see that there is no substitution θ' such that

$$\{\alpha \leqslant \texttt{Real}, \beta \leqslant \texttt{Real}, \alpha \leqslant \gamma, \beta \leqslant \gamma\} \vdash \{\theta'(\delta) \leqslant \texttt{Real}, \theta'(\delta \to \delta \to \delta) \leqslant \alpha \to \beta \to \gamma\}$$

so

$$\varnothing; \delta \leqslant \texttt{Real} \vdash \texttt{max} : \delta \to \delta \to \delta \quad \not\ll \quad \varnothing; C \vdash \texttt{max} : \alpha \to \beta \to \gamma$$

in other words, although we know that the two typings for `max` are equivalent over the class of Helly posets, the halbstark relation is too strong to validate the equivalence. Of course we expect this: the $\vdash$ relation between constraint sets uses syntactic entailment and so is independent of the choice of underlying order, but as we have seen, the simplification is not valid over a 2-crown. Thus we introduce a weaker instance relation on typing judgements, which exploits our notion of semantic entailment.

3.2.1 Definition: $\mathcal{D}_1 \equiv \Gamma_1; C_1 \;\vdash\; e : \tau_1$ has as an instance $\mathcal{D}_2 \equiv \Gamma_2; C_2 \;\vdash\; e : \tau_2$ (we write $\mathcal{D}_1 \lll \mathcal{D}_2$) if there is a substitution θ such that

1. $C_2 \models^{\mathcal{Q}}_{\overline{\alpha}} \theta(C_1) \cup \{\theta\tau_1 \leqslant \tau_2\}$

2. $\Gamma_2 = \theta(\Gamma_1)$

where $\overline{\alpha}$ is the set of variables occurring in τ_2 and Γ_2.

It is hard to see how this instance relation can be shown to be sound in the syntactic sense of [22]: that if we have a typing derivation of $\mathcal{D}_1$ and $\mathcal{D}_1 \lll \mathcal{D}_2$ then we have a typing derivation of $\mathcal{D}_2$; rather we would need some semantic notion of the soundness of an instance relation. There are, however, good reasons to adhere to syntactic soundness: we can expect compilation to be defined on elaborated terms produced by type inference, so simplification on constraint sets had better have an operational interpretation on the corresponding elaborated terms. For example, the derivation

$$\varnothing; C \;\vdash\; \mathtt{max} : \alpha{\rightarrow}\beta{\rightarrow}\gamma$$

can be interpreted as building, in the presence of the coercions $c_1{:}\alpha{\rightarrow}\mathtt{Real}$, $c_2{:}\beta{\rightarrow}\mathtt{Real}$, $c_3{:}\alpha{\rightarrow}\gamma$, $c_4{:}\beta{\rightarrow}\gamma$, the elaborated term

$$\lambda x{:}\alpha.\lambda y{:}\beta.\mathtt{if}\; c_1(x) < c_2(y)\; \mathtt{then}\; c_3(x)\; \mathtt{else}\; c_4(y)$$

The statement

$$\varnothing; C \;\vdash\; \mathtt{max} : \alpha{\rightarrow}\beta{\rightarrow}\gamma \quad\ll\quad \varnothing; \delta{\leqslant}\mathtt{Real} \;\vdash\; \mathtt{max} : \delta{\rightarrow}\delta{\rightarrow}\delta$$

then constructs, via the substitution $[\delta/\alpha] \circ [\delta/\beta] \circ [\delta/\gamma]$ the coercions $c_5{:}\delta{\rightarrow}\mathtt{Real}$ and $c_6{:}\delta{\rightarrow}\delta$, and the term

$$\lambda x{:}\delta.\lambda y{:}\delta.\mathtt{if}\; c_5(x) < c_5(y)\; \mathtt{then}\; c_6(x)\; \mathtt{else}\; c_6(y)$$

This observation extends to simplification: if we decrease the size of the coercion set for a term, it is necessary to be able to build a term elaborated with appropriate coercions chosen from the reduced constraint set. If the simplified constrained type is a halbstark instance of the original (and we have the substitution and proof terms to justify this) then we can always do so, and in this sense simplification of a judgement to one of its halbstark instances is constructive simplification. It is harder to see how we might operationally interpret simplification to a semantic instance. However, we will demonstrate a practical, constructive approach to simplification based on our semantic entailment relation. When simplifying $\mathcal{D}_1$ to $\mathcal{D}_2$ we shall ensure always that

1. $\mathcal{D}_1 \ll \mathcal{D}_2$, and we shall be able to derive the necessary substitution and proofs which justify entailment

2. $\mathcal{D}_2 \lll \mathcal{D}_1$

Syntactic soundness will ensure that simplification has the required operational nature; semantic completeness that no possible solution will be lost in any possible context where the constraints might be instantiated. In this setting we can simplify `max`, because with the substitution $[\delta/\gamma] \circ [\delta/\beta] \circ [\delta/\gamma]$ we obtain

$$\delta{\leqslant}\texttt{Real} \;\vdash\; \delta{\leqslant}\delta,\; \delta{\leqslant}\delta,\; \delta{\leqslant}\texttt{Real},\; \delta \leqslant \texttt{Real}, \delta{\to}\delta{\to}\delta{\leqslant}\delta{\to}\delta{\to}\delta$$

and using the empty substitution for semantic entailment,

$$\alpha{\leqslant}\gamma,\; \beta{\leqslant}\gamma,\; \alpha{\leqslant}\texttt{Real},\; \beta{\leqslant}\texttt{Real},\; \models^{\mathcal{Q}}_{\{\alpha,\beta,\gamma\}} \delta{\leqslant}\texttt{Real},\; \delta{\to}\delta{\to}\delta{\leqslant}\alpha{\to}\beta{\to}\gamma$$

over any class of Helly posets $\mathcal{Q}$, since in any solution θ for the LHS, we can set δ to be the least upper bound of $\theta\alpha$ and $\theta\beta$, which exists in any Helly poset by prop(2.2.8).

Our strategy for simplification will operate on the set of distance inequalities which are the closure of the set of constraints. We will define a simplifying transformation, *H-contraction* on sets of distance inequalities, which will reduce distances whilst preserving semantic equivalence, and from a set of distance inequalities thus contracted we will extract a simplifying substitution and additional constraints which generate a halbstark instance of the original constraint set. There is a slight difference in the notion of completeness when dealing with constrained type schemes rather than just constraint sets: in the latter case we must preserve all solutions, whereas in the former we need only preserve those ground instances of the type scheme which are minimal (amongst ground instances); then for any type τ at which we intended to use the original type scheme we are guaranteed to have an instance of the simplified type which can be coerced to τ.

Our simplification algorithm is based on an entailment relation, and we will choose the simplest: entailment over the class of Helly posets. So tautness is simply closure under Dual, Join, and Min. We will also assume that our set of distance inequalities is connected, since if it is not, we simplify it component-wise.

Contraction

Suppose that $\mathcal{I}$ is C's closure under Dual, Symm, and Join. For variables constrained by C is semantically the same as to be constrained by $\mathcal{I}$, so we will consider $\Gamma; \mathcal{I} \ \vdash \ e : \tau$ to have the obvious meaning, and to be semantically equivalent to $\Gamma; C \ \vdash \ e : \tau$.

Our simplification algorithm operates by reducing the distances in $\mathcal{I}$. Suppose we have a derivation $\Gamma; \mathcal{I} \ \vdash \ e : \tau$, and we wish to add $d(\tau_1, \tau_2) \leqslant e$ to $\mathcal{I}$ to obtain $\mathcal{I}'$ such that $\Gamma; \mathcal{I}' \ \vdash \ e : \tau$ is semantically equivalent. Let $\theta = [\overline{\beta}/\overline{\alpha}]$ be a renaming substitution on the internal variables in $\mathcal{I}'$ using fresh variables $\overline{\beta}$; then the conditions we require are that

$$\theta\mathcal{I}' \models \mathcal{I} \cup \{\tau \leqslant \theta\tau\}$$

and

$$\mathcal{I} \models \theta\mathcal{I}' \cup \{\theta\tau \leqslant \tau\}$$

for the variables in Γ and τ over the class of Helly posets which retract to the underlying base poset. Since every solution of $\mathcal{I}'$ is a solution of $\mathcal{I}$, the first is immediate. So writing $\mathcal{J} = \theta\mathcal{I}' \cup \{\theta\tau \leqslant \tau\}$, our condition is equivalent to requiring the distances in $\mathcal{I}$ between observables to be less than (or equal to) the distances in $\mathcal{J}$. In fact, for observables α_1 and α_2, we must have

$$d_{\mathcal{J}}(\alpha_1, \alpha_2) = d_{\mathcal{J}}(\alpha_1, \theta\alpha_1) + d_{\mathcal{J}}(\theta\alpha_1, \theta\alpha_2) + d_{\mathcal{J}}(\alpha_2, \theta\alpha_2)$$

since for any observable α, either α occurs in τ, in which case the only path from an α to any other atom in $\mathcal{J}$ is via $\theta(\alpha)$, or $\alpha = \theta\alpha$, so $d(\alpha, \theta\alpha) = \langle 1, 1\rangle$. And since θ is just a renaming on $\mathcal{I}$,

$$d_{\mathcal{J}}(\theta\alpha_1, \theta\alpha_2) = d_{\theta\mathcal{I}}(\theta\alpha_1, \theta\alpha_2) = d_{\mathcal{I}}(\alpha_1, \alpha_2)$$

Now since for any variable α occurring in τ the only inequalities relating α and $\theta\alpha$ are derived from the decomposition of the inequality $d(\theta\tau, \tau) \leqslant \langle 1, 2\rangle$, by structural induction on the type τ

- $d_{\mathcal{J}}(\alpha, \theta\alpha) = \langle 2, 1\rangle$ if α occurs only positively in τ
- $d_{\mathcal{J}}(\alpha, \theta\alpha) = \langle 1, 2\rangle$ if α occurs only negatively in τ
- $d_{\mathcal{J}}(\alpha, \theta\alpha) = \langle 1, 1\rangle$ otherwise.

So choosing $\langle k,k\rangle$ larger than any distance in $\mathcal{J}$, we define the *polarity function* ζ on the atoms occurring in $\mathcal{I}$:

$$\begin{array}{rcll}\zeta(\alpha) & = & \langle 2,1\rangle & \text{if } \alpha \text{ occurs only positively in } \tau\\ \zeta(\alpha) & = & \langle 1,2\rangle & \text{if } \alpha \text{ occurs only negatively in } \tau\\ \zeta(\alpha) & = & \langle k,k\rangle & \text{if } \alpha \text{ is an internal}\\ \zeta(\alpha) & = & \langle 1,1\rangle & \text{otherwise}\end{array}$$

so that $\zeta(\alpha) = d_{\mathcal{J}}(\alpha, \theta\alpha)$ for each external α, and our condition is equivalent to requiring for every pair of atoms α_1 and α_2 occurring in $\mathcal{I}$ that

$$d_{\mathcal{I}}(\alpha_1,\alpha_2) \leqslant \zeta(\alpha_1) + d_{\mathcal{J}}(\theta\alpha_1,\theta\alpha_2) + \zeta(\alpha_2)^{-1}$$

We can now define a transformation on sets of distance inequalities:

3.2.2 Definition [Contraction]: Suppose $\mathcal{I}$ is a set of distance inequalities over atomic types with a polarity function ζ, and τ_1 and τ_2 are types occurring in $\mathcal{I}$. $d(\tau_1,\tau_2) \leqslant e$ is a *contraction* of $\mathcal{I}$ if

- $d_{\mathcal{I}}(\tau_1,\tau_2) \nleqslant e$
- for any α_1,α_2 occurring in $\mathcal{I}$

$$d_{\mathcal{I}}(\alpha_1,\alpha_2) \leqslant \zeta(\alpha_1) + d_{\mathcal{I}}(\alpha_1,\tau_1) + e + d_{\mathcal{I}}(\tau_2,\alpha_2) + \zeta(\alpha_2)^{-1}$$

Although it is convenient to use the condition $d_{\mathcal{I}}(\tau_1,\tau_2) \leqslant e$, obviously the contraction decreases the distance from τ_1 to τ_2 to $e \wedge d_{\mathcal{I}}(\tau_1,\tau_2)$. A contraction is thus exactly any reduction of a single distance inequality in $\mathcal{I}$ which preserves semantic equivalence.

Verifying that an inequality is a contraction requires time quadratic in the size of the constraint set. Thus we introduce a transformation which equally effective but easier to validate.

3.2.3 Definition [H-contraction]: Suppose $\mathcal{I}$ is a set of distance inequalities over atomic types with a polarity function ζ, and τ_1 and τ_2 are types occurring in $\mathcal{I}$. $d(\tau_1,\tau_2) \leqslant e$ is a *H-contraction* of $\mathcal{I}$ if

- $d_{\mathcal{I}}(\tau_1,\tau_2) \nleqslant e$
- for every τ, $d_{\mathcal{I}}(\tau,\tau_2) \leqslant \zeta(\tau) + d_{\mathcal{I}}(\tau,\tau_1) + e$

3.2.4 Proposition: If $d(\tau_1,\tau_2) \leqslant e$ is an H-contraction of $\mathcal{I}$, it is a contraction of $\mathcal{I}$.

Proof: Let α_1, α_2 be atoms occurring in $\mathcal{I}$, and $\mathcal{J} = \mathcal{I} \cup \{d(\tau_1, \tau_2) \leqslant e$. We will show that for any pair of atoms α_1, α_2 occurring in $\mathcal{I}$,

$$d_{\mathcal{I}}(\alpha_1, \alpha_2) \leqslant \zeta(\alpha_1) + d_{\mathcal{J}}(\alpha_1, \alpha_2) + \zeta(\alpha_2)^{-1}$$

which is equivalent to showing that for any chain X of minimal length from α_1 to α_2,

$$d_{\mathcal{I}}(\alpha_1, \alpha_2) \leqslant \zeta(\alpha_1) + e_X + \zeta(\alpha_2)^{-1}$$

where e_X is the length of X. So suppose X is a chain of minimal length. If X does not contain a link from τ_1 to τ_2 or vice versa, then the result follows immediately from the fact that X is a chain in $\mathcal{I}$. Otherwise, since X is of minimal length, we may assume the link occurs at most once in the chain. So suppose X contains a link from τ_1 to τ_2, then the length of X is at least

$$d_{\mathcal{I}}(\alpha_1, \tau_1) + e + d_{\mathcal{I}}(\tau_2, \alpha_2)$$

So

$$\begin{array}{rcl} \zeta(\alpha_1) + e_X + \zeta(\alpha_2)^{-1} & \geqslant & \zeta(\alpha_1) + d_{\mathcal{I}}(\alpha_1, \tau_1) + e + d_{\mathcal{I}}(\tau_2, \alpha_2) + \zeta(\alpha_2)^{-1} \\ & \geqslant & d_{\mathcal{I}}(\alpha_1, \tau_2) + d_{\mathcal{I}}(\tau_2, \alpha_2) + \zeta(\alpha_2)^{-1} \\ & & \quad \text{since } d(\tau_1, \tau_2) \leqslant e \text{ is a H-contraction} \\ & \geqslant & d_{\mathcal{I}}(\alpha_1, \alpha_2) \end{array}$$

The case where X has a link is from τ_2 to τ_1 is dual. □

Although not every contraction is a H-contraction, we can use H-contractions to obtain exactly the same distance reductions as can be performed with contractions.

3.2.5 Proposition: Suppose no H-contraction is possible on $\mathcal{I}$. Then no contraction is possible.

Proof: Suppose no H-contraction is possible, and we have $d(\tau_1, \tau_2) \leqslant e$ with $d_{\mathcal{I}}(\tau_1, \tau_2) \nleqslant e$. Since $d(\tau_1, \tau_2) \leqslant e$ is not a H-contraction, there is some α_1 such that

$$d_{\mathcal{I}}(\alpha_1, \tau_2) \nleqslant \zeta(\alpha_1) + d_{\mathcal{I}}(\alpha_1, \tau_1) + e$$

Thus

$$d_{\mathcal{I}}(\tau_2, \alpha_1) \nleqslant e^{-1} + d_{\mathcal{I}}(\tau_1, \alpha_1) + \zeta(\alpha_1)^{-1}$$

But $d_{\mathcal{I}}(\tau_2, \alpha_1) \leqslant e^{-1} + d_{\mathcal{I}}(\tau_1, \alpha_1) + \zeta(\alpha_1)^{-1}$ is not a H-contraction, so there must be some α_2 such that

$$d_{\mathcal{I}}(\alpha_2, \alpha_1) \nleqslant \zeta(\alpha_2) + d_{\mathcal{I}}(\alpha_2, \tau_2) + e^{-1} + d_{\mathcal{I}}(\tau_1, \alpha_1) + \zeta(\alpha_1)^{-1}$$

But then

$$d_{\mathcal{I}}(\alpha_1, \alpha_2) \not\leqslant \zeta(\alpha_1) + d_{\mathcal{I}}(\alpha_1, \tau_1) + e + d_{\mathcal{I}}(\tau_2, \alpha_2) + \zeta(\alpha_2)^{-1}$$

so $d(\tau_1, \tau_2) \leqslant e$ is not a contraction. □

From Distances to Constraints

Having completely H-contracted a set of distance inequalities, it remains to simplify the constraint set from the set of minimal distances. Our strategy will be as follows: If we obtain a minimal distance of $\langle 1, 1\rangle$ between two types, then they must have the same value in any solution, and we can identify them in the constraint set by substitution. If we obtain $\langle 1, 2\rangle$ or $\langle 2, 1\rangle$, we can choose to add this to the constraint set if its transitive reduction is thereby decreased. Since adding these distance inequalities to $\mathcal{I}$ preserves the possible minimal typings of $\forall\overline{\alpha}\backslash\mathcal{I}.\,\tau$, adding them to C will do the same. Thus we have:

3.2.6 Proposition: Suppose we have a derivation $\mathcal{D}_1 \equiv \Gamma \ \vdash \ e : \forall\overline{\alpha}\backslash C.\,\tau$, and using H-contraction we obtain the substitution θ and additional constraint set C'. Let $\mathcal{D}_2 \equiv \theta\Gamma \ \vdash \ e : \forall\overline{\alpha}\backslash\theta C \cup C'.\,\theta\tau$. Then $\mathcal{D}_1 \ll \mathcal{D}_2$, and $\mathcal{D}_2 \lll \mathcal{D}_1$.

Proof: It is easy to see that substitution and weakening of constraint sets results in a halbstark instance.

From the discussion of H-contraction, it should be clear that for any ground instance θ_g of the $\mathcal{D}_1$, θ_g is a solution of $\theta C \cup C'$, and we have $(\theta_g \circ \theta)\tau \leqslant \theta\tau$. Thus $\mathcal{D}_2 \lll \mathcal{D}_1$. □

3.2.7 Remark: If $\mathcal{I}$ cannot be contracted (or equivalently, cannot be H-contracted), then any reduction of any distance inequality results in set in which some solution is lost. Thus if a type scheme has a constraint-free equivalent representation, i.e. one in which all distances have been contracted to $\langle 1, 1\rangle$, it will be found by repeated H-contraction.

Comparison with other approaches

While we have only considered this approach in the context of atomic types, in principle simplification techniques over Helly posets have wider applicability than model-based approaches which operate over lattices only. A fairly direct comparison is possible with the system of Fuh and Mishra [18], also based on atomic constraint sets, which uses syntactic rather than semantic criteria for simplification. They define two kinds of transformation on constraint sets: *S-simplification* and *G-simplification.*

3.2.8 Definition: Let $\uparrow_C(\alpha)$ be the set of atoms α' such that $\alpha \leqslant \alpha'$ is in the transitive closure of C, and $\downarrow_C(\alpha)$ the set of atoms α' such that $\alpha' \leqslant \alpha$ correspondingly.

A G-simplification on a constraint set C is a substitution $[\alpha_2/\alpha_1]$ where α_1 is an internal variable, and $\uparrow_C(\alpha_1)\backslash\{\alpha\} \subseteq \uparrow_C(\alpha_2)$ and $\downarrow_C(\alpha_1)\backslash\{\alpha\} \subseteq \downarrow_C(\alpha_2)$. G-minimisation is similar, but uses multi-point substitutions which generate provably equivalent constraint sets.

An S-simplification is a substitution $[\alpha_2/\alpha_1]$ where either

- α_1 is an external occurring only positively such that $\alpha_2 \in \uparrow_C(\alpha_1)$, and $\uparrow_C(\alpha_1)\backslash\{\alpha_1\} \subseteq \uparrow_C(\alpha_2)$
- α_1 is an external occurring only negatively such that $\alpha_2 \in \downarrow_C(\alpha_1)$, and $\downarrow_C(\alpha_1)\backslash\{\alpha_1\} \subseteq \downarrow_C(\alpha_2)$

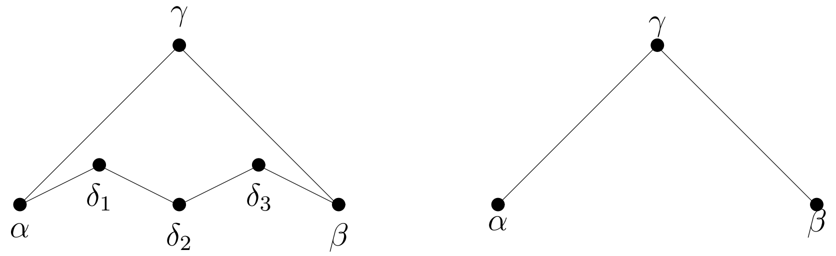
In fact, this version of S-simplification is weaker than Fuh and Mishra's in that it operates on external variables in τ but not in Γ. Fuh and Mishra use a slightly different form of instance relation which allows subtyping on contexts, for example, a term variable bound in Γ with type `Nat` can be replaced with a binding to `Int`. Whilst weakening the hypotheses in Γ in this way results in a stronger judgement, on partial derivations (where later we may for example apply f:`Nat`→`Int` to x, it is not sound.

Substitution of α_1 for α_2 is equivalent to asserting that $d(\alpha_1, \alpha_2) = \langle 1, 1\rangle$, and it is not hard to see that the form of S-simplification defined here is thus a special cases of H-contraction. In fact G-minimisation can be expressed as a sequence of H-contractions: If θ is a multipoint substitution on C which constitutes a G-minimisation, since θC is provably equivalent to C, it has the same solution set. And since the combined substitution does not reduce the solution set, neither does any of the single point substitutions of which it consists: thus they are all H-contractions.

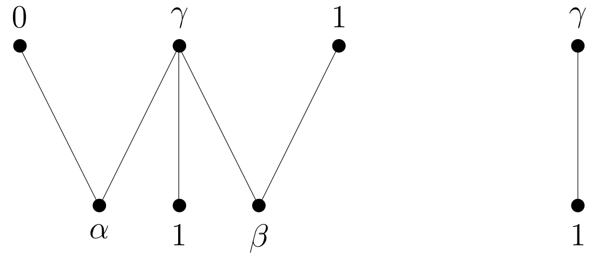
However, since it operates over a narrower class of posets, we would expect H-contraction to be capable of transformations which cannot be derived by S-simplification and G-minimisation, and we will demonstrate that this is indeed the case.

3.3 Examples

Simplifying constraint sets is the special case of H-contraction where $\zeta(\alpha) = \langle 1, 1\rangle$ for every observable α. Under these circumstances, a H-contraction is just a contraction which does not change the minimal distances between pairs of observables. And if the contraction is to a distance of $\langle 1, 1 \rangle$, it is equivalent to a substitution. The following two examples of problem constraint sets are taken from [43].

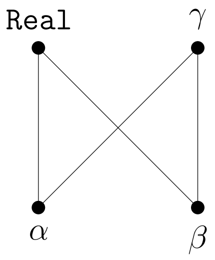


Here α, β, and γ are external and δ_1, δ_2, and δ_3 internal. While G-minimisation simplifies the constraint set on the left immediately to that on the right by the multi-point substitution $[\gamma/\delta_1] \circ [\gamma/\delta_2] \circ [\gamma/\delta_3]$, there is no single-point substitution which satisfies the criteria for being a G-simplification. On the other hand none of the substitutions (applied in any order) $[\gamma/\delta_1]$, $[\gamma/\delta_2]$, $[\gamma/\delta_3]$ change any of the minimal distances between α, β, and γ, so we obtain the simplification as a sequence of H-contractions. Notice that by a different sequence of H-contractions we could obtain (for example) the simplifying substitution $[\alpha/\delta_1] \circ [\alpha/\delta_2] \circ [\gamma/\delta_3]$.



Here the constraint set on the left has external γ, and internals α and β, and base types $0 < 1$. It is shown in [43] that this type cannot be simplified by G-minimisation and S-simplification, but neither substitution in the sequence $[0/\alpha] \circ [1/\beta]$ changes any of the minimal distances, so both are H-contractions.

Our third example, this time of a constrained type, is `max`:



Here we have

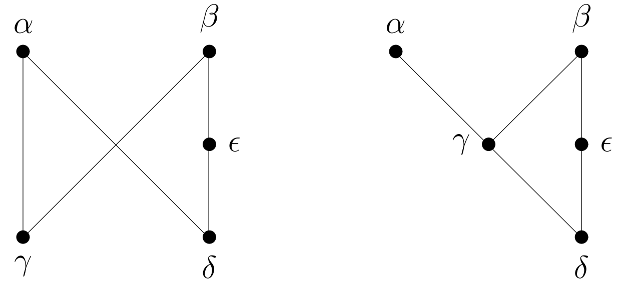
$$\begin{array}{lcl}
\zeta(\alpha) & = & \langle 1,2\rangle \\
\zeta(\beta) & = & \langle 1,2\rangle \\
\zeta(\gamma) & = & \langle 2,1\rangle \\
\zeta(\texttt{Real}) & = & \langle 1,1\rangle
\end{array}$$

We can verify the requirement for H-contracting $d(\alpha,\beta)$ from $\langle 2,3\rangle$ to $\langle 2,1\rangle$, that for each τ, $d(\tau,\beta) \leqslant \zeta(\tau) + d_{\mathcal{I}}(\tau,\alpha) + \langle 2,1\rangle$.

$$\begin{array}{lclclcl}
d(\alpha,\beta) & \leqslant & \langle 1,2\rangle & + & \langle 1,1\rangle & + & \langle 2,1\rangle \\
d(\beta,\beta) & \leqslant & \langle 1,1\rangle & + & \langle 2,3\rangle & + & \langle 2,1\rangle \\
d(\gamma,\beta) & \leqslant & \langle 1,2\rangle & + & \langle 2,1\rangle & + & \langle 2,1\rangle \\
d(\texttt{Real},\beta) & \leqslant & \langle 1,1\rangle & + & \langle 2,1\rangle & + & \langle 2,1\rangle
\end{array}$$

So this is a valid H-contaction. An informal explanation of this transformation is that since any solution θ is over a Helly poset, $\theta\alpha$ and $\theta\beta$ have an upper bound which is a lower bound for γ and `Real`. If we were to insert a fresh variable ϵ into the constraint set with this property, then we would obtain the substitution $[\epsilon/\alpha]$ as an S-simplification. In any case, the remaining simplifying substitutions $[\alpha/\gamma]$ and $[\alpha/\beta]$ are just H-contractions equivalent to S-simplifications.

It should be mentioned that it is not always the case that H-contraction can be usefully exploited to simplify types. For example, in the following case, with all variables external and $\zeta(\gamma) = \langle 1,2\rangle$, $\zeta(\alpha) = \zeta(\beta) = \zeta(\delta) = \zeta(\epsilon) = \langle 1,1\rangle$ the constraint set on the left, where γ is internal, can be H-contracted to that on the right, but the number of constraints does not decrease.



3.4 Tautness over Smaller Classes

Tautness over Lattices

We will show that over lattices, to obtain tautness, we need only apply the obvious rules:

$$\frac{\mathcal{I} \vdash d(\tau_1,\tau_2) \leqslant e}{\mathcal{I} \vdash d(\tau_1,\tau_2) \leqslant \langle 2,2\rangle} \qquad (\textsc{Truncate})$$

$$\frac{\mathcal{I} \vdash d(\alpha, \mathtt{a}) \leqslant \langle 1,2\rangle \qquad \mathcal{I} \vdash d(\alpha, \mathtt{b}) \leqslant \langle 1,2\rangle}{\mathcal{I} \vdash d(\alpha, \mathtt{a} \wedge \mathtt{b}) \leqslant \langle 1,2\rangle} \tag{\textsc{Inf}}$$

$$\frac{\mathcal{I} \vdash d(\alpha, \mathtt{a}) \leqslant \langle 2,1\rangle \qquad \mathcal{I} \vdash d(\alpha, \mathtt{b}) \leqslant \langle 2,1\rangle}{\mathcal{I} \vdash d(\alpha, \mathtt{a} \vee \mathtt{b}) \leqslant \langle 2,1\rangle} \tag{\textsc{Sup}}$$

3.4.1 Proposition [Soundness & Completeness of Deductive Closure]: If $\mathcal{J}$ is the closure of $\mathcal{I}$ by Dual, Min, Join, Inf, Sup, and Truncate. then for any lattice Q' which is a Q-model, θ is a solution of $\mathcal{I}$ iff θ is a solution of $\mathcal{J}$.

Proof: Straightforward. □

3.4.2 Proposition [Tautness of Deductive Closure]: Let Q be a lattice, and $\mathcal{I}$ be a solvable set of distance inequalities between variables and elements of Q. If $\mathcal{I}$ is closed under Dual, Min, Join, Inf, Sup, and Truncate then $\mathcal{I}$ is taut over the class of lattices which retract to Q.

Proof: We will proceed by demonstrating that if $d_{\mathcal{I}}(\tau_1, \tau_2) = e$, then there is a Q-model Q' and a solution θ such that $d_{Q'}(\theta\tau_1, \theta\tau_2) = e$. We will ensure the equality holds by adding sufficient inequalities to $\mathcal{I}$ and demonstrating that the resulting set of inequalities is still distance consistent. We proceed by cases on e:

$\langle 1,1\rangle$ is immediate

$\langle 1,2\rangle$ We must have $\mathrm{U}(\tau_1) \leqslant \mathrm{U}(\tau_2)$, since for every $\mathtt{a} \geqslant \tau_1$ we also have $\mathtt{a} \geqslant \tau_2$. If $\mathrm{U}(\tau_1) < \mathrm{U}(\tau_2)$ then the distance equality holds exactly in the maximal solution. If $\mathrm{U}(\tau_1) = \mathrm{U}(\tau_2)$, we must have $\mathrm{D}(\tau_1) < \mathrm{U}(\tau_1)$, since otherwise we have $\tau_1 \geqslant \mathrm{D}(\tau_1) = \mathrm{U}(\tau_1) \geqslant \tau_2$, contradicting the assumption that $d_{\mathcal{I}}(\tau_1, \tau_2) = \langle 1,2\rangle$. Thus we can choose some $\mathtt{a}$ such that $\mathrm{D}(\tau_1) \leqslant \mathtt{a} < \mathrm{U}(\tau_1)$, and set $\mathcal{I}' = \mathcal{I} \cup \{\tau_1 \leqslant \mathtt{a}\}$. Now if $\mathcal{I}'$ is not consistent, there must be some chain not in $\mathcal{I}$ which is not in $\mathcal{I}$, and thus must include $d(\tau, \mathtt{a}) \leqslant \langle 1,2\rangle$ as a link; indeed the shortest such chain must start or end with such a link. Suppose the latter. Then we must have some $\mathtt{b} \leqslant \tau_1$ such that $\mathtt{b} \nleqslant \mathtt{a}$, which cannot occur since $\mathrm{D}(\tau_1) \leqslant \mathtt{a}$. Now the required inequality holds in the maximal solution of $\mathcal{I}'$.

$\langle 2,1\rangle$ is dual to the above case

$\langle 2,2\rangle$ Here we cannot have $\mathrm{U}(\tau_1) \leqslant \mathrm{D}(\tau_2)$ or $\mathrm{U}(\tau_2) \leqslant \mathrm{D}(\tau_1)$ by the assumption that $d_{\mathcal{I}}(\tau_1, \tau_2) = \langle 2,2\rangle$. We set $Q' = Q \cup \{\mathtt{c}_1, \mathtt{c}_2\}$ with the inequalities

$$\mathrm{D}(\tau_i) < \mathtt{c}_i < \mathrm{U}(\tau_i)$$

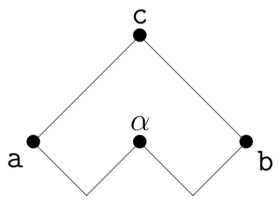
It is straightforward that this is a lattice and a Q-model via the retraction $r(\mathtt{c}_i) = \mathrm{U}(\tau_i)$, and that $d_{Q'}(\mathtt{c}_1, \mathtt{c}_2) = \langle 2, 2\rangle$. By a similar argument to the previous case

$$\mathcal{I} \cup \{d(\tau_1, \mathtt{c}_1) = \langle 1, 1\rangle, d(\tau_2, \mathtt{c}_2) = \langle 1, 1\rangle\}$$

is distance consistent over Q', and so there is a solution θ such that $d_{Q'}(\theta\tau_1, \theta\tau_2) = \langle 2, 2\rangle$

Tautness over Trees

Trees have two interesting features from the point of view of computing tautenings: firstly, every tree contains a top element, hence the minimal distance between two elements is at most $\langle 2, 3\rangle$. Secondly, any two elements in a tree which share a lower bound are comparable: i.e. if $d(\alpha, \beta) \leqslant \langle 3, 2\rangle$, then for any solution θ either $\theta\alpha \leqslant \theta\beta$ or $\theta\alpha \geqslant \theta\beta$. So if we have the following situation:



where $\mathtt{a}$ and $\mathtt{b}$ have no common lower bound, we may deduce that since α is comparable with both $\mathtt{a}$ and $\mathtt{b}$, we must have $\alpha \geqslant \mathtt{c}$.

It turns out that these two observations are sufficient to characterise trees. We introduce two rules which formalise these observations.

$$\frac{\mathcal{I} \;\vdash\; d(\tau_1, \tau_2) \leqslant e}{\mathcal{I} \;\vdash\; d(\tau_1, \tau_2) \leqslant \langle 2, 3\rangle} \qquad (\textsc{Truncate})$$

$$\frac{\mathcal{I} \;\vdash\; d(\tau, \mathtt{a}) \leqslant \langle 3, 2\rangle \qquad \mathcal{I} \;\vdash\; d(\tau, \mathtt{b}) \leqslant \langle 3, 2\rangle \quad \mathtt{a}, \mathtt{b} \text{ have no common lower bound}}{\mathcal{I} \;\vdash\; d(\tau, \mathtt{a} \vee \mathtt{b}) \leqslant \langle 2, 1\rangle} \qquad (\textsc{NoInf})$$

3.4.3 Proposition [Soundness & Completeness of Deductive Closure]: Let Q be a tree, and $\mathcal{I}$ some set of distance inequalities over variables and elements of Q. Suppose $\mathcal{I}'$ is the deductive closure of $\mathcal{I}$ under Dual, Join, Min, Truncate, and NoInf. Then for any tree Q' which is a Q-model, $\mathcal{I}'$ and $\mathcal{I}'$ are equivalent over Q'.

Proof: For every pair of variables or base types, the minimal distance from one to the other will be at least as short in $\mathcal{I}'$ as in $\mathcal{I}$. So if a substitution satisfies $\mathcal{I}'$, it also satisfies $\mathcal{I}$.

Conversely let θ be a solution of $\mathcal{I}$ over Q'. It is straightforward that the solution satisfies those distance inequalities generated by Truncate and NoInf.

□

3.4.4 Proposition [Tautness of Deductive Closure]: Suppose $\mathcal{I}$ is a set of distance inequalities between variables and atomic types over a tree Q. If $\mathcal{I}$ is deductively closed under Dual, Join, Min, Truncate, and NoInf, then $\mathcal{I}$ is taut over the class of trees Q' which are Q-models.

Proof: Suppose $\mathcal{I}$ is closed under these rules, τ_1 and τ_2 are types occurring in $\mathcal{I}$, and $d_{\mathcal{I}}(\tau_1, \tau_2) \leqslant e$. We will demonstrate tautness by adding suitable inequalities to $\mathcal{I}$ and choosing a suitable extension of Q to ensure either than this distance holds exactly, or in the case where e is $\langle 2, 2\rangle$, that both the distances $\langle 1, 2\rangle$ and $\langle 2, 1\rangle$ can be made to hold exactly. We will proceed by cases on e: note that over a tree the minimal distance between two elements is at most $\langle 2, 3\rangle$.

$\langle 1, 1\rangle$ Any substitution θ which solves $\mathcal{I}$ must have $\theta\tau_1 = \theta\tau_2$.

$\langle 1, 2\rangle$ It suffices to find a solution θ such that $\theta\tau_1 < \theta\tau_2$. Let $\mathtt{b}_0 = \mathrm{U}(\tau_2)$. Note that $\mathcal{I} \nvdash \tau_1 \geqslant \mathtt{b}_0$, since otherwise we would have $\mathcal{I} \vdash d(\tau_1, \tau_2) = \langle 1, 1\rangle$ by Min. There is at most one child $\mathtt{b}_1$ of $\mathtt{b}_0$ such that $d(\tau_1, \mathtt{b}_1) \leqslant \langle 3, 2\rangle$, since if there were more than one we would have $d(\tau_1, \mathtt{b}_0) \leqslant \langle 2, 1\rangle$ by NoInf. If such a $\mathtt{b}_1$ exists, we set $Q' = Q$, else $Q' = Q \cup \{\mathtt{b}_1 \leqslant \mathtt{b}_0\}$ for $\mathtt{b}_1$ fresh, and in either case set $\mathcal{I}' = \mathcal{I} \cup \{\tau_1 \leqslant \mathtt{b}_1\}$.

We need to show $\mathcal{I}'$ is solvable over Q', which we will derive from the distance consistency of $\mathcal{I}$. Suppose we have $d_{\mathcal{I}'}(\mathtt{c}, \mathtt{d}) < d_{\mathcal{I}}(\mathtt{c}, \mathtt{d})$. By induction on a minimal chain in $\mathcal{I}'$ from $\mathtt{c}$ to $\mathtt{d}$ we must have a chain from $\mathtt{b}_1$ to either $\mathtt{c}$ (we will suppose $\mathtt{c}$) or $\mathtt{d}$ which is shorter in $\mathcal{I}'$ than in $\mathcal{I}$, whose first link is from $\mathtt{b}_1$ to τ_1 (and thus of length $\langle 2, 1\rangle$), and whose other links are all in $\mathcal{I}$. Thus it will suffice to show that for any $\mathtt{c}$, $d_{\mathcal{I}'}(\mathtt{b}_1, \mathtt{c}) = d_{\mathcal{I}}(\mathtt{b}_1, \mathtt{c})$. We will proceed by cases:

- if $\mathtt{c} \geqslant \mathtt{b}_0$ we have $d_{\mathcal{I}}(\tau_1, \mathtt{c}) = d_{\mathcal{I}}(\mathtt{b}_1, \mathtt{c}) = \langle 1, 2\rangle$. And $\langle 2, 1\rangle + \langle 1, 2\rangle \geqslant \langle 1, 2\rangle$.
- if $\mathtt{c} \leqslant \mathtt{b}_1$, then $d_{\mathcal{I}}(\mathtt{b}_1, \mathtt{c})$ is either $\langle 1, 1\rangle$ or $\langle 2, 1\rangle$, and in either case $\langle 2, 1\rangle + d_{\mathcal{I}}(\tau_1, \mathtt{c}) \geqslant d_{\mathcal{I}}(\mathtt{b}_1, \mathtt{c})$.

- if $\mathtt{c}$ is incomparable with $\mathtt{b}_0$, then we cannot have $d_{\mathcal{I}}(\tau_1, \mathtt{c}) \leqslant \langle 2, 3\rangle$, since then, as we have $\mathcal{I} \ \vdash \ d(\tau_1, \mathtt{b}_0) \leqslant \langle 3, 2\rangle$, we would have $\mathcal{I} \ \vdash \ \tau_1 > \mathtt{b}_0 \wedge c$ by NoInf. Thus the minimal distance in $\mathcal{I}'$ from $\mathtt{b}_1$ to $\mathtt{c}$ via τ_1 is at least $\langle 2, 3\rangle$.
- if $\mathtt{c} < \mathtt{b}_0$ but $\mathtt{c} \not< \mathtt{b}_1$ and $d_{\mathcal{I}}(\tau_1, \mathtt{c}) \leqslant \langle 3, 2\rangle$ we have $\mathcal{I} \ \vdash \ \tau_1 > \mathtt{b}_0$ by NoInf. Thus again the minimal distance in $\mathcal{I}'$ from $\mathtt{b}_1$ to $\mathtt{c}$ via τ_1 is at least $\langle 2, 3\rangle$.

In all cases we have $d_{\mathcal{I}'}(\mathtt{b}_1, \mathtt{c}) = d_{\mathcal{I}}(\mathtt{b}_1, \mathtt{c})$.

$\langle 2, 1\rangle$ If $d(\tau_1, \tau_2) = \langle 2, 1\rangle$, then $d(\tau_1, \tau_2) = \langle 1, 2\rangle$, so the construction is as above.

$\langle 2, 2\rangle$ Notice it is impossible to produce a solution in which $d_Q(\theta\tau_1, \theta\tau_2) = \langle 2, 2\rangle$. Instead we demonstrate the existence of solutions in which $d_Q(\theta\tau_1, \theta\tau_2) = \langle 1, 2\rangle$ and $d_Q(\theta\tau_1, \theta\tau_2) = \langle 2, 1\rangle$. Again we define $\mathtt{b}_0 = \mathrm{U}(\tau_2)$, and again we cannot have $\mathcal{I} \ \vdash \ \tau_1 \geqslant \mathtt{b}_0$, since otherwise we would have $\mathcal{I} \ \vdash \ d(\tau_1, \tau_2) \leqslant \langle 2, 1\rangle$ by Join. Now if $\mathcal{I}'$ is as before, its distance consistency follows by exactly the same argument as the $\langle 1, 2\rangle$ case.

$\langle 2, 3\rangle$ (i.e. $\theta\tau_1$, $\theta\tau_2$ are incomparable). We define

$$\mathtt{b}_0 = \wedge\{\mathtt{b} \mid d_{\mathcal{I}}(\mathtt{b}, \tau_1) \leqslant \langle 3, 2\rangle \text{ and } d_{\mathcal{I}}(\mathtt{b}, \tau_2) \leqslant \langle 3, 2\rangle\}$$

This set is non-empty since $\top$ is in the set, and down-closed: if there are incomparable $\mathtt{c}_1$ and $\mathtt{c}_2$, then for each of $d_{\mathcal{I}}(\mathtt{c}_1, \tau_1) \leqslant \langle 3, 2\rangle$ and $d_{\mathcal{I}}(\mathtt{c}_2, \tau_1) \leqslant \langle 3, 2\rangle$ so $\tau_1 \geqslant \mathtt{c}_1 \wedge \mathtt{c}_2$ by NoInf, which together with $\tau_1 \leqslant \mathtt{c}_1$ and $\tau_1 \leqslant \mathtt{c}_2$ implies $\mathtt{c}_1 = \mathtt{c}_2$. so the two cannot be incomparable. Again we cannot have $\mathcal{I} \ \vdash \ \tau_1 \geqslant \mathtt{b}_0$ or $\mathcal{I} \ \vdash \ \tau_2 \geqslant \mathtt{b}_0$. Now there is at most one child $\mathtt{b}_1$ of $\mathtt{b}_0$ satisfying $d_{\mathcal{I}}(\mathtt{b}_1, \tau_1) \leqslant \langle 2, 3\rangle$, and one satisfying $d_{\mathcal{I}}(\mathtt{b}_2, \tau_2) \leqslant \langle 2, 3\rangle$, if either of these children do not exist we set $Q' = Q \cup \{\mathtt{b}_1, \mathtt{b}_2\}$ (as required) and $\mathcal{I}' = \mathcal{I} \cup \{\tau_1 \leqslant \mathtt{b}_1, \tau_2 \leqslant \mathtt{b}_2\}$. Note that $\mathtt{b}_1$ and $\mathtt{b}_2$ are guaranteed distinct.

Again we can show the distance consistency of $\mathcal{I}'$ from that of $\mathcal{I}$: if $\mathcal{I}$ is not distance consistent there must be some down-fence from $\mathtt{b}_1$ (or $\mathtt{b}_2$) via τ_1 (or τ_2) which is shorter in $\mathcal{I}'$ than $\mathcal{I}$ and whose other links are all in $\mathcal{I}$. Again the reasoning is essentially the same as the $\langle 1, 2\rangle$ case for each chain from $\mathtt{b}_1$ via τ_1 or $\mathtt{b}_2$ via τ_2. □

3.4.5 Remark: In fact, the proof of tautness only requires the ability to extend trees at their leaf nodes. Since tautness over any class of posets implies tautness over any superclass, the result is equally applicable to tautness over the class of trees generated by adding new children to existing nodes, for example single inheritance hierarchies.

3.5 Conclusion

The technique presented here can be seen as specialising techniques for simplification over the class of all partial orders such as those of [18], to Helly posets. We have seen that our system is more powerful, and the restriction on possible orderings seems a reasonable tradeoff for better simplification. It would be interesting to investigate further properties of contraction, particularly in respect to completeness and confluence.

Since we have a theory about entailment relations over particular classes of posets such as lattices and trees, it should be possible to design more powerful simplification algorithms which exploit the fact that these relations verify more equivalences.

Although we envisage simplification occurring only prior to `let`-bindings, an additional refinement might be to simplify previous `let`-bound types as more information about distances between common λ-bound variables is discovered.

Chapter 4

Solvability and Solution

Although we have seen how systems of constraints can be tested for solvability and solved over atomic Helly posets, in practice this is not a realistic approach to constraint sets obtained by type inference, which may well contain constraints of the form e.g. $\alpha \leqslant \tau_1 \to \tau_2$. Although if we are dealing with finite types and structural subtyping it is possible (subject to a simple unification-like check for circularity) to reduce the constraint set to atomic form with substitutions of the form e.g. $[\alpha_1 \to \alpha_2 / \tau]$, this results in a potentially exponential increase in the number of terms in the constraint set, and moreover when working with non-structural subtyping the reduction is less straightforward, and in the case of regular types, is not available at all.

In this chapter we will consider the issues of working with posets of regular types built with type constructors from some base Helly poset. We will generalise the notion of simulation to allow us to test the distance relation in such orderings, and show how sets of constraints over the orderings may be efficiently tested for solvability and solved. We will treat base types as constructors of arity 0 throughout.

4.1 Decomposition and Simulation

In chapter 1 it was shown how the subtype relation on regular types can be characterised as a maximal simulation: that is, the maximal relation consistent with $\sqsubseteq$ and closed under certain propagation rules. Thus to test the subtype relation between two types, it suffices to find a simulation between them. The simulation which justifies the $\leqslant$ relation has two components: a *shallow approximation* relation between type constructors, and rules for propagating the simulation to the constructor arguments. The propagated inequalities must be necessary for the original inequality to hold, and, together with the shallow approximation, sufficient. We can demonstrate similar properties for the distance relation.

It is straightforward to construct from our shallow approximation $\sqsubseteq$ on constructors a shallow approximation of the distance relation:

$$\begin{array}{lcll}
d_Q(c,c) & = & \langle 1,1\rangle & \text{for any constructor } c \\
d_Q(\mathtt{a},\mathtt{b}) & = & d_Q(\mathtt{a},\mathtt{b}) & \text{for all } \mathtt{a},\mathtt{b} \in Q \\
d_Q(\{\}_I,\{\}_J) & = & \langle 1,2\rangle & \text{if } I \supset J \\
d_Q(\{\}_I,\{\}_J) & = & \langle 2,1\rangle & \text{if } I \subset J \\
d_Q(\{\}_I,\{\}_J) & = & \langle 2,2\rangle & \text{otherwise} \\
d_Q(c,\top) & = & \langle 1,2\rangle & \text{for any } c \neq \top \\
d_Q(\bot,c) & = & \langle 1,2\rangle & \text{for any } c \neq \bot
\end{array}$$

and in order to propagate the constructor inequalities to their arguments, we introduce the following notion:

4.1.1 Definition: An ordering is *decomposable* if for any $\tau = c_1\tau_1\ldots\tau_m$ and $\rho = c_2\rho_1\ldots\rho_n$ (perhaps containing free variables) and any $e \in \mathcal{L}$ such that $d(c_1,c_2) \leqslant e$, there is a *decomposition set* $D(\rho,\tau,e)$ where every element of D is of the form (ρ',τ',e') and such that

- every ρ' is either one of the ρ_i or a ground type, and every τ' is either one of the τ_i or a ground type
- each τ_i and ρ_i occurs at most once, and if both occur, then they occur in the same tuple
- for any substitution θ on the variables in ρ and τ, $d_Q(\theta\rho,\theta\tau) \leqslant e$ iff $d_Q(\theta\rho',\theta\tau') \leqslant e'$ for every $(\rho',\tau',e') \in D(\rho,\tau,e)$.

If τ_1 and τ_2 are both ground types, then any substitution is just the identity on τ_1 and τ_2. So then

- if $d(\tau_1,\tau_2) \leqslant e$, each of the distance inequalities in $D(\tau_1,\tau_2,e)$ holds
- if $d(\nu(\tau_1),\nu(\tau_2)) \leqslant e$ then $d(\tau_1,\tau_2) \leqslant e$ if each of the distance inequalities in $D(\tau_1,\tau_2,e)$ holds.

4.1.2 Definition: A set S of tuples (τ_1,τ_2,e) is *closed under decomposition* if for any $(\tau_1,\tau_2,e) \in Z$, $d(\nu(\tau_1),\nu(\tau_2)) \leqslant e$ and $D(\tau_1,\tau_2,e) \subseteq Z$. For $e_0 \in \mathcal{L}$ a e_0*-simulation* $Z(\tau_1,\tau_2,e_0)$ between two regular types τ_1 and τ_2 is a finite set of tuples containing (τ_1,τ_2,e_0) which is closed under decomposition.

We will use graph representations of types to build simulations, and read the existence of a vertex $((G_1, r_1), (G_2, r_2), e)$ in the simulation as a requirement that $d_Q((G_1, m), (G_2, n)) \leqslant e$. It is easy to see that this notion of simulation corresponds with the definition for the $\leqslant$ relation in the case $e = \langle 1, 2\rangle$, where $((G_1, m), (G_2, n), \langle 1, 2\rangle)$ corresponds with $(G_1, m) \sqsubseteq (G_2, n)$ and $((G_1, m), (G_2, n), \langle 2, 1\rangle)$ with $(G_2, n) \sqsubseteq (G_1, m)$

Structural Constructors

Let Q_0 be Helly a poset of base types, and Q built from Q_0 with structural constructors. Suppose we have a distance inequality $d(\mathtt{T}\tau_1 \dots \tau_n, \mathtt{T}\tau'_1 \dots \tau'_n) \leqslant e$. Since constructor types are only comparable with other types headed by the same constructor, the points making up the fences which generate the distance e must also be of the form $\mathtt{T}\rho_1 \dots \rho_n$. So there is an up-fence in Q from $\mathtt{T}\tau_1 \dots \tau_n$ to $\mathtt{T}\tau'_1 \dots \tau'_n$ of length r exactly if

- for each i such that $p^{\mathtt{T}}_i = \mathtt{pos}$, there is an up-fence in Q from τ_i to τ'_i of length n.
- for each i such that $p^{\mathtt{T}}_i = \mathtt{neg}$, there is a down-fence in Q from τ_i to τ'_i of length n
- for each i such that $p^{\mathtt{T}}_i = \mathtt{mix}$, $\tau_i = \tau'_i$.

Thus

$$\begin{array}{rcl} D(\mathtt{T}\tau_1 \dots \tau_k, \mathtt{T}\tau'_1 \dots \tau'_k, e) & = & \{(\tau_i, \tau'_i, e) \mid p^i_{\mathtt{T}} = \mathtt{pos}\} \\ & \cup & \{(\tau_i, \tau'_i, e^*) \mid p^i_{\mathtt{T}} = \mathtt{neg}\} \\ & \cup & \{(\tau_i, \tau'_i, \langle 1, 1\rangle) \mid p^i_{\mathtt{T}} = \mathtt{mix}\} \end{array}$$

4.1.3 Proposition: Let Q be the order defined by the base types in Q_0 and the $\to$-constructor rule. There is a simulation $Z(\tau_1, \tau_2, e_0)$ iff $d_Q(\tau_1, \tau_2) \leqslant e_0$.

Proof: Suppose $d_Q(\tau_1, \tau_2) \leqslant e_0$, and (G_1, r_1) and (G_2, r_2) are graphs of τ_1 and τ_2. Let Z be the minimal set containing $((G_1, r_1), (G_2, r_2), e_0)$, and for every $((G_1, m), (G_2, n), e) \in Z$ with $\nu(m) = \nu(n) = \to$ $((G_1, m(1)), (G_2, m(1)), e^*) \in Z$, and $((G_1, m(2)), (G_2, m(2)), e) \in Z$, i.e. the minimal closure of $\{((G_1, r_1), (G_2, r_2), e_0)\}$ under decomposition. It is straightforward to demonstrate that Z is a simulation.

Conversely, suppose there is a simulation Z. In order to prove that for every $((G_1, m), (G_2, n), e) \in Z$, $d((G_1, m), (G_2, n)) \leqslant e$, we will construct types on the up-fences and down-fences which make up this distance. For every vertex $z = ((G_1, m), (G_2, n), \langle u, d\rangle)$ of Z we define an up-fence $U(z)$ and a down-fence $D(z)$ of length u and d respectively, labelled with type constructors as follows:

- if $\nu(m) = \nu(n) = \rightarrow$, then $\nu(U(z)(i)) = \nu(D(z)(i)) = \rightarrow$ for all i
- if $\nu(m) = \mathtt{a}, \nu(n) = \mathtt{b}$, so that $d_Q(\mathtt{a}, \mathtt{b}) \leqslant \langle u, d\rangle$ and there is an up-fence of length u and a down fence of length d from a to b, then $\nu(U(z)(i))$ is the ith point on the up-fence, and $\nu(D(z)(i))$ the ith point on the down-fence.

So for any $z \in Z$, if i is even, then $\nu(U(x)(i)) \sqsubseteq \nu(U(x)(i+1))$ for any x and $\nu(D(x)(i+1)) \sqsubseteq \nu(D(x)(i))$, and similarly if i is odd, $\nu(U(x)(i+1)) \sqsubseteq \nu(U(x)(i))$ and $\nu(D(x)(i)) \sqsubseteq \nu(D(x)(i+1))$.
We add transitions

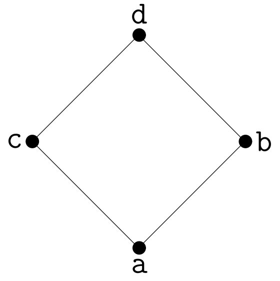
$$\begin{array}{rcl} U(m,n,e)(i) & \overset{1}{\rightarrow} & D(m(1), n(1), e^*)(i) \\ U(m,n,e)(i) & \overset{2}{\rightarrow} & U(m(2), n(2), e)(i) \\ D(m,n,e)(i) & \overset{1}{\rightarrow} & U(m(1), n(1), e^*)(i) \\ D(m,n,e)(i) & \overset{2}{\rightarrow} & D(m(1), n(1), e)(i) \end{array}$$

to form a transition graph G_Z. Now we can define the fences which make up the distance e_0 from τ_1 to τ_2. Let $z_0 = ((G_1, r_1), r_2, e_0)$. Let $u_i = U(z_0)(i)$, and $d_i = D(z_0)(i)$. Then it is easy to show by induction that if i is even, there is a simulation between the graphs (G_Z, u_i) and (G_Z, u_{i+1}), and if odd, between (G_Z, u_{i+1}) and (G_Z, u_i), and dually for the points d_i. Notice that

$$\begin{array}{rcccl} (G_Z, U(z_0)(0)) & = & (G_Z, D(z_0)(0)) & \cong & (G_1, r_1) \\ (G_Z, U(z_0)(u)) & = & (G_Z, D(z_0)(d)) & \cong & (G_2, r_2) \end{array}$$

and so the graphs rooted at (G_Z, u_i) and (G_Z, d_i) form the required up- and down-fences to generate the distance e_0. □

4.1.4 Example: In order to illustrate this construction, we will take a simple case the underlying order Q given by

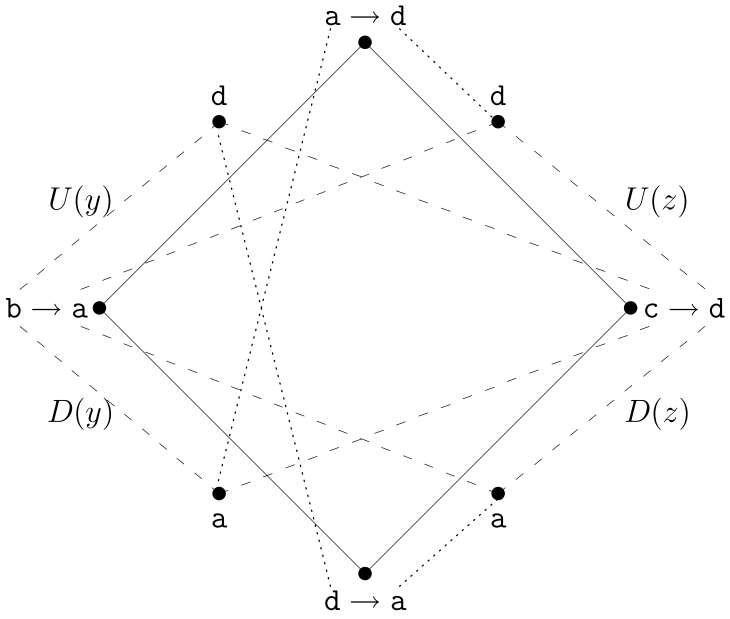


and the inequality $d(\mathtt{b}\rightarrow\mathtt{a}, \mathtt{c}\rightarrow\mathtt{d}) \leqslant \langle 2, 2\rangle$. Defining

$$\begin{array}{rcl} x & = & (\mathtt{b}\rightarrow\mathtt{a}, \mathtt{c}\rightarrow\mathtt{d}, \langle 2, 2\rangle) \\ y & = & (\mathtt{b}, \mathtt{c}, \langle 2, 2\rangle) \\ z & = & (\mathtt{a}, \mathtt{d}, \langle 2, 2\rangle) \end{array}$$

the simulation is simply $\{x, y, z\}$ and we take

$$\begin{array}{rcll}
U(x) & = & \to, \to, \to & \\
D(x) & = & \to, \to, \to & \\
U(y) & = & \mathtt{b}, \mathtt{d}, \mathtt{c} & \text{an up-fence from } \mathtt{b} \text{ to } \mathtt{c} \\
D(y) & = & \mathtt{b}, \mathtt{a}, \mathtt{c} & \text{a down-fence from } \mathtt{b} \text{ to } \mathtt{c} \\
U(z) & = & \mathtt{a}, \mathtt{d}, \mathtt{d} & \text{an up-fence from } \mathtt{a} \text{ to } \mathtt{d} \\
D(z) & = & \mathtt{a}, \mathtt{a}, \mathtt{d} & \text{a down-fence from } \mathtt{a} \text{ to } \mathtt{d}
\end{array}$$



(solid lines are inequalities, dashed lines are inferred fences, dotted lines indicate the source of the derived types).

Observe that since → is contravariant in its first argument, the first argument of the type of $U(x)(1)$ comes from $D(y)(1)$, and the first argument of $D(x)(1)$ is drawn from $U(y)(1)$. Then the point u_1 on the up-fence from $\mathtt{b}{\to}\mathtt{a}$ to $\mathtt{c}{\to}\mathtt{d}$ is $D(y)(1) \to U(z)(1) = \mathtt{a}{\to}\mathtt{d}$, and the point d_1 on the down-fence is $U(y)(1) \to D(z)(1) = \mathtt{d}{\to}\mathtt{a}$.

Record Types

If we add a record constructor, the situation becomes more interesting. When $e < \langle 2, 2\rangle$, for any $d(\{\}_I.\{\}_J) \leqslant e$ we inherit from the rules for the $\leqslant$-constructor,

$$D(\{l_i{:}\tau_i\}_{i\in I}, \{l_j{:}\tau'_j\}_{j\in J}, e) = \{(\tau_j, \tau'_j, e) \mid j \in I \cap J\}$$

For longer distances, notice that any two records have an upper bound, so that the decomposition set for $e \geqslant \langle 2, 3\rangle$ is empty. This leaves the cases $e = \langle 2, 2\rangle$ and

$e = \langle 3, 2\rangle$. If two record types have a common lower bound, then the subterms for any common label must also have a lower bound, e.g suppose we have the distance inequality $d(\{l{:}\tau_1, m{:}\tau_2\}, \{l{:}\tau_1', n{:}\tau_2'\}) \leqslant \langle 2, 2\rangle$, then the two records have a lower bound. Now any lower bound must have fields labelled l, m, and n, and the type of the l field must be a common lower bound for τ_1 and τ_1', so we require $d(\tau_1, \tau_1') \leqslant \langle 3, 2\rangle$. Observe that no such deductions can be made about τ_2 and τ_2'. Thus if $\langle 2, 2\rangle \leqslant e \leqslant \langle 3, 2\rangle$,

$$D(\{l_i{:}\tau_i\}_{i\in I}, \{l_j{:}\tau_j'\}_{j\in J}, e) = \{(\tau_i, \tau_i', \langle 3, 2\rangle) \mid i \in I \cap J\}$$

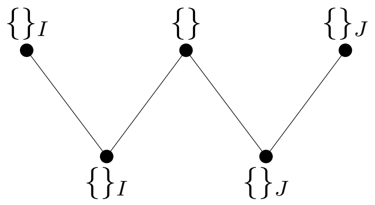
4.1.5 Proposition: Let Q be the order defined by the base types in Q_0 and the $\rightarrow$ and $\{\}$ constructor rules. If there is a simulation $Z(\tau_1, \tau_2, e_0)$, then $d_Q(\tau_1, \tau_2) \leqslant e_0$.

Proof: The proof extends the proof for $\rightarrow$ in a fairly straightforward fashion. We define U and D for vertices (m, n, e) where $\nu(m) = \{\}_I$ and $\nu(n) = \{\}_J$ and $e = \langle u, d\rangle$ as follows:

$$\begin{array}{llcll}
u = 1 & U(m,n,e) & = & (\{\}_I, \{\}_J) & \\
u \geqslant 2 & U(m,n,e) & = & (\{\}_I, \{\}, \{\}, \ldots \{\}, \{\}_J) & u \text{ even} \\
 & & & (\{\}_I, \{\}, \{\}, \ldots \{\}, \{\}_J, \{\}_J) & u \text{ odd} \\
d = 1 & D(m,n,e) & = & (\{\}_I, \{\}_J) & \\
d = 2 & D(m,n,e) & = & (\{\}_I, \{\}_{I\cup J}, \{\}_J) & \\
d \geqslant 3 & D(m,n,e) & = & (\{\}_I, \{\}_I, \{\}, \ldots \{\}, \{\}_J, \{\}_J) & d \text{ even} \\
 & & & (\{\}_I, \{\}, \{\}, \ldots \{\}, \{\}_J) & d \text{ odd}
\end{array}$$

observe that $U(m, n, \langle u, d\rangle)$ is an up-fence of constructors of length u, and $D(m, n, \langle u, d\rangle)$ is an up-fence of constructors of length d.

There is also a further necessary extension of the construction: suppose we have $d((G_1, r_1), (G_2, r_2)) \leqslant \langle 3, 4\rangle$, where $r_1 = \{\}_I$, and $r_2 = \{\}_J$. Then the simulation is just $\{(G_1, r_1), (G_2, r_2), \langle 3, 4\rangle\}$. $D(z_0)$ is then



when we build the down fence between (G_1, r_1) and (G_2, r_2), the constructors $\{\}_I$ and $\{\}_J$ must be given out-edges, which unlike the case of $\rightarrow$ cannot be constructed from the fences between constructor arguments, because the constructor arguments do not occur in the simulation. In fact, the distance inequality will hold irrespective of the arguments we use, so the only constraints upon their construction are the boundary conditions, that

$$\begin{array}{rclcl}(G_Z, U(z_0)(0)) & = & (G_Z, D(z_0)(0)) & \cong & (G_1, r_1)\\ (G_Z, U(z_0)(u)) & = & (G_Z, D(z_0)(d)) & \cong & (G_2, r_2)\end{array}$$

The obvious solution is simply to augment the construction with the argument vertices in the graphs G_1 and G_2 themselves. In this case, for each $i \in I$, we will have $D(z_0)(0) \xrightarrow{i} (r_1(i))$ $D(z_0)(1) \xrightarrow{i} (r_1(i))$ and for each $j \in J$, $D(z_0)(3) \xrightarrow{j} (r_2(j))$ and $D(z_0)(4) \xrightarrow{j} (r_2(j))$, and the constructed types now form the required down-fence.

It remains to define the transitions which generate the simulations. Suppose we have a vertex $(m, n, \langle u, d\rangle) \in Z$

$$\begin{array}{lllll}
u = 1 & U(m,n,e)(0) & \xrightarrow{l} & U(m(l),n(l),e)(0) & \text{for each } l \in M\\
 & U(m,n,e)(1) & \xrightarrow{l} & U(m(l),n(l),e)(1) & \text{for each } l \in N\\
 & & & & \\
u \geqslant 2 & U(m,n,e)(0) & \xrightarrow{l} & m(l) & \text{for each } l \in M\\
 & U(m,n,e)(u-1) & \xrightarrow{l} & n(l) & \text{for each } l \in N\text{, if } u \text{ odd}\\
 & U(m,n,e)(u) & \xrightarrow{l} & n(l) & \text{for each } l \in N\\
 & & & & \\
d = 1 & D(m,n,e)(0) & \xrightarrow{l} & D(m(l),n(l),e)(0) & \text{for each } l \in M\\
 & D(m,n,e)(1) & \xrightarrow{l} & D(m(l),n(l),e)(1) & \text{for each } l \in N\\
 & & & & \\
d = 2 & D(m,n,e)(0) & \xrightarrow{l} & D(m(l),n(l),e)(0) & \text{for each } l \in M\\
 & D(m,n,e)(1) & \xrightarrow{l} & D(m(l),n(l),e)(0) & \text{for each } l \in M - N\\
 & & \xrightarrow{l} & D(m(l),n(l),e)(1) & \text{for each } l \in M \cap N\\
 & & \xrightarrow{l} & D(m(l),n(l),e)(2) & \text{for each } l \in N - M\\
 & D(m,n,e)(2) & \xrightarrow{l} & D(m(l),n(l),e)(2) & \text{for each } l \in N\\
 & & & & \\
d \geqslant 3 & D(m,n,e)(0) & \xrightarrow{l} & m(l) & \text{for each } l \in M\\
 & D(m,n,e)(1) & \xrightarrow{l} & m(l) & \text{for each } l \in M\\
 & D(m,n,e)(d-1) & \xrightarrow{l} & n(l) & \text{for each } l \in N\text{, if } d \text{ even}\\
 & D(m,n,e)(d) & \xrightarrow{l} & n(l) & \text{for each } l \in N
\end{array}$$

The rest of the proof goes through as before. □

⊤ and ⊥

Introducing ⊤ and ⊥ into the subtyping order has a more wide-ranging effect. Now any two types are an upper and a lower bound (⊤ and ⊥ respectively) and so the decomposition set for any inequality of distance $\langle 2,2\rangle$ or greater is empty. For smaller distances the decomposition sets are unchanged. It is quite straightforward to extend propositions(4.1.3) and (4.1.5) to deal with these constructors.

Nonatomic base types

We can also use this technique to deal with nonatomic base types. We will consider the case of → constructors only. Suppose we have an inequality of the form $d(\mathtt{a}, \tau_1{\rightarrow}\tau_2) \leqslant e$, where $\Sigma^{\Uparrow}(\mathtt{a})$ is an →-type, and suppose the minimal distance from $\mathtt{a}$ to $\tau_1{\rightarrow}\tau_2$ is $\langle u, d\rangle$. Since any fence from $\mathtt{a}$ to $\tau_1{\rightarrow}\tau_2$ must pass through $\Sigma^{\Uparrow}(\mathtt{a})$, the shortest down-fence of length d must have the form $\mathtt{a} \geqslant \mathtt{a}' \leqslant \tau_1'{\rightarrow}\tau_2' \geqslant \ldots$ for some $\mathtt{a}' \leqslant \mathtt{a}$, where $\Sigma^{\Uparrow}(\mathtt{a}) \leqslant \tau_1'{\rightarrow}\tau_2'$. So there will be an up-fence $\Sigma^{\Uparrow}(\mathtt{a}) \leqslant \tau_1'{\rightarrow}\tau_2' \geqslant \ldots$ from $\Sigma^{\Uparrow}(\mathtt{a})$ to $\tau_1{\rightarrow}\tau_2$ of length $d-1$. Thus we know that $d(\mathtt{a}, \tau_1{\rightarrow}\tau_2) \leqslant e$ iff $d(\Sigma^{\Uparrow}(\mathtt{a}), \tau_1{\rightarrow}\tau_2) \leqslant e^{\vartriangle}$, and so $D(\mathtt{a}, \tau, e) = D(\Sigma^{\Uparrow}(\mathtt{a}), \tau, e^{\vartriangle})$.

So if $\mathtt{a}$ is a non-atomic base type and c is not a base type, our shallow approximation extends as follows:

$$\begin{aligned} d(\mathtt{a}, c) &= d(\nu(\Sigma^{\Uparrow}(\mathtt{a})), c)^{\vartriangle} \\ d(c, \mathtt{a}) &= d(c, \nu(\Sigma^{\Uparrow}(\mathtt{a})))^{\blacktriangle} \end{aligned}$$

And our decomposition sets, if $\Sigma^{\Uparrow}(\mathtt{a}) = \rho_1{\rightarrow}\rho_2$ are

$$\begin{aligned} D(\mathtt{a}, \tau_1{\rightarrow}\tau_2, e) &= D(\Sigma^{\Uparrow}(\mathtt{a}), \tau_1{\rightarrow}\tau_2, e^{\vartriangle}) &= \{(\rho_1, \tau_1, (e^{\vartriangle})^*), (\rho_2, \tau_2, e^{\vartriangle})\} \\ D(\tau_1{\rightarrow}\tau_2, \mathtt{a}, e) &= D(\tau_1{\rightarrow}\tau_2, \Sigma^{\Uparrow}(\mathtt{a}), e^{\blacktriangle}) &= \{(\tau_1, \rho_1, (e^{\blacktriangle})^*), (\tau_2, \rho_2, e^{\blacktriangle})\} \end{aligned}$$

4.2 Solvability and Solution

Closure and Consistency

Let Q be a decomposable ordering. A distance inequality $d(\tau_1, \tau_2) \leqslant e$ possibly containing type variables is *Q-consistent* if at least one of τ_1 or τ_2 is a variable, or $d(\nu(c_1), \nu(c_2)) \leqslant e$. A set of distance inequalities is *Q-consistent* if every member is Q-consistent.

4.2.1 Definition [Closure]: If Q is decomposable ordering, $\mathcal{I}$ is *Q-closed* if $\mathcal{I}$ is closed under Dual, Min, and Join, the rule Truncate

$$\frac{d(\rho, \tau) \leqslant e}{d(\rho, \tau) \leqslant e \wedge \delta_Q} \qquad (\textsc{Truncate})$$

and the rule Decomp

$$\frac{d(\rho, \tau) \leqslant e \quad d_Q(\nu(\rho), \nu(\tau)) \leqslant e \quad (\rho', \tau', e') \in D(\rho, \tau, e)}{d(\rho', \tau') \leqslant e'} \qquad (\textsc{Decomp})$$

The *Q-closure* of $\mathcal{I}$ $\mathrm{close}_Q(\mathcal{I})$, is the minimal Q-closed superset of $\mathcal{I}$.

We include the rule Truncate because it ensures that the closure is finite, since it can only contain types occurring in $\mathcal{I}$ and distances at most δ_Q.

4.2.2 Proposition: Let Q be a decomposable partial order of diameter δ_Q, $\mathcal{I}$ a set of distance inequalities, and $\mathcal{J}$ be the Q-closure of $\mathcal{I}$. Then any solution of $\mathcal{I}$ is a solution of $\mathcal{J}$ and vice versa.

Proof: Let θ be a solution of $\mathcal{J}$. Since for any τ_1 and τ_2, $d_{\mathcal{J}}(\tau_1, \tau_2) \leqslant d_{\mathcal{I}}(\tau_2, \tau_2)$, θ is a solution of $\mathcal{I}$. Suppose conversely that θ is a solution of $\mathcal{I}$. We proceed by induction on the height of the derivation an inequality $d_Q(\theta\tau_1, \theta\tau_2) \leqslant e$ in $\mathcal{J}$. The last rule in deriving this from the distance inequalities in $\mathcal{I}$ is either Min, Dual, Join, Truncate, or Decomp. For each rule if θ satisfies the premises, then θ also satisfies the conclusion. Thus since θ is a solution of $\mathcal{I}$ over Q, it is a solution of $\mathcal{J}$. □

Clearly if the Q-closure of $\mathcal{I}$ is inconsistent, then $\mathcal{I}$ is unsolvable. Our fundamental result will be that under certain quite general conditions, a set of constraints is solvable iff its Q-closure is Q-consistent. Before we prove this result, a few examples will demonstrate the use of distance closure, particularly its advantage over the usual notion of constraint closure.

With structural subtyping over the tree order $\texttt{Long} < \texttt{Int}, \texttt{Nat} < \texttt{Int}$, the constraint set $\{\texttt{Nat}{\to}\texttt{Nat} \leqslant \alpha, \texttt{Long}{\to}\texttt{Long} \leqslant \alpha\}$ is unsolvable even though transitive and subterm closure of constraints yields no inconsistencies. Distance closure enables us to deduce $d(\texttt{Long}, \texttt{Nat}) \leqslant \langle 3, 2\rangle$:

$$\frac{d(\texttt{Nat}{\to}\texttt{Nat}, \alpha) \leqslant \langle 1, 2\rangle \qquad \dfrac{d(\texttt{Long}{\to}\texttt{Long}, \alpha) \leqslant \langle 1, 2\rangle}{d(\alpha, \texttt{Long}{\to}\texttt{Long}) \leqslant \langle 2, 1\rangle}}{\dfrac{d(\texttt{Nat}{\to}\texttt{Nat}, \texttt{Long}{\to}\texttt{Long}) \leqslant \langle 2, 3\rangle}{d(\texttt{Nat}, \texttt{Long}) \leqslant \langle 3, 2\rangle}}$$

Note that although $d(\texttt{Long}, \texttt{Nat}) \leqslant \langle 3, 2\rangle$ is enough to demonstrate inconsistency, we can deduce the stronger inequality $d(\texttt{Long}, \texttt{Nat}) \leqslant \langle 2, 2\rangle$.
Similarly from $\{\{l{:}\texttt{a}\} \geqslant \alpha, \{l{:}\texttt{b}\} \geqslant \alpha\}$ we can deduce $d(\texttt{a}, \texttt{b}) \leqslant \langle 3, 2\rangle$.
The reason that constraint closure is sufficient over a lattice is that in a lattice the only distances of any interest are $\langle 1, 1\rangle$, $\langle 1, 2\rangle$, and $\langle 2, 1\rangle$. All elements in a component are at most $\langle 2, 2\rangle$ apart, so beyond limiting a type to a particular component, a distance inequality of length $\langle 2, 2\rangle$ or greater provides no information. Thus if the ordering is just one big lattice, or if shape matching can be used to decompose the constraint set into separate components, all remaining information can be expressed as constraints.

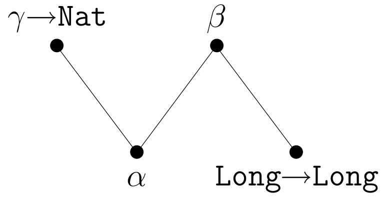
Constructing a Solution

In this section we will demonstrate a quite general method for constructing solutions of constraint sets over subtype orderings of regular types. Our technique can be seen as an extension of that of [33], using distance inequalities instead of constraints.

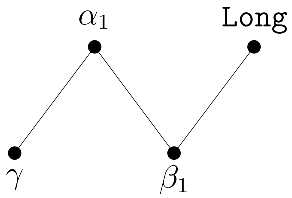
We will consider first a simple example: an ordering formed by → over the poset $Q = \{\texttt{Long}\leqslant\texttt{Int}, \texttt{Nat}\leqslant\texttt{Int}\}$. Consider how we might go about constructing a solution to the constraint set

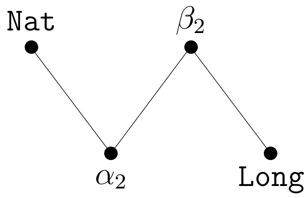
$$C = \{\alpha\leqslant\gamma\rightarrow\texttt{Nat}, \alpha\leqslant\beta, \texttt{Long}\rightarrow\texttt{Long}\leqslant\beta, \gamma\leqslant\texttt{Nat}\}$$

$\mathcal{I}$, the closure of C has two components, one of which is

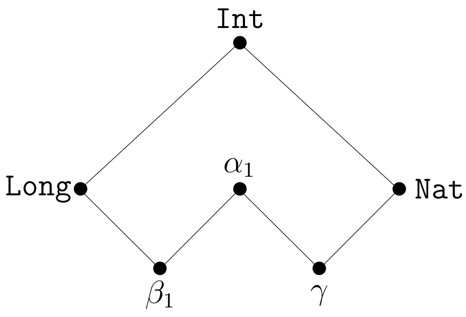


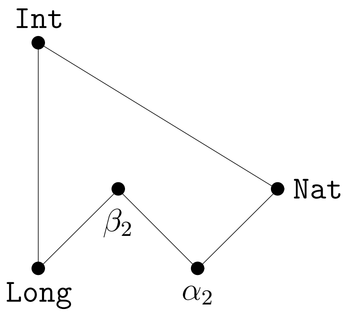
It's easy to see that α and β must be → types (clearly they cannot be atomic types), say $\alpha_1\rightarrow\alpha_2$ and $\beta_1\rightarrow\beta_2$. Propagating the above constraints to the constructor arguments we obtain





Combining these with the inequalities in Q_0 and the inequality $\gamma\leqslant\texttt{Nat}$ (and with a little cosmetic reshaping of the diagrams) we obtain:





Applying the HSF $f(\alpha) = \bigwedge\{\texttt{a} \mid \texttt{a} \geqslant \alpha\}$ of proposition(2.4.3), we obtain the solution

$$\{\alpha_1 = \texttt{Int}, \alpha_2 = \texttt{Nat}, \beta_1 = \texttt{Long}, \beta_2 = \texttt{Int}, \gamma = \texttt{Nat}\}$$

Notice however, that we could have solved for each of α, and β, independently using distance inequalities: the points labelled with other variables become "anonymous". For example, we can obtain

$$\begin{array}{lcl} d(\alpha, \texttt{Long}{\rightarrow}\texttt{Long}) & \leqslant & \langle 2,3\rangle \\ d(\alpha, \gamma{\rightarrow}\texttt{Nat}) & \leqslant & \langle 1,2\rangle \end{array}$$

by distance closure from the original constraint set, then having determined the substitution $[\alpha_1{\rightarrow}\alpha_2/\alpha]$, we obtain

$$\begin{array}{lcl} d(\alpha_1, \gamma) & \leqslant & \langle 2,1\rangle \\ d(\alpha_1, \texttt{Long}) & \leqslant & \langle 3,2\rangle \end{array}$$

then by closure with the original constraint set and Q_0 we obtain

$$\begin{array}{lcl} d(\alpha_1, \texttt{Nat}) & = & \langle 3,2\rangle \\ d(\alpha_1, \texttt{Long}) & = & \langle 3,2\rangle \\ d(\alpha_1, \texttt{Int}) & = & \langle 3,2\rangle \end{array}$$

whence we can apply the HSF. We can consider the process of solving for a variable α to have three steps:

choice decide on a constructor c for a variable (e.g. α) using the minimal distances from α to all the constructor types.

decomposition generate a set of inequalities on each argument to c, using the decomposition rules

$\mathcal{I}$-expansion close this set of inequalities with those in $\mathcal{I}$ to obtain the full set of conditions on each argument

If we are going to solve for α, β, and γ independently, we need to find a function g for choosing constructors which ensure that the types we build do indeed form a solution. In particular, if we have two variables α and β such that e.g. $d(\alpha, \beta) \leqslant \langle 1,2\rangle$, we require that $g(\alpha) \sqsubseteq g(\beta)$, and that we generate sufficient conditions on the arguments of $g(\alpha)$ and $g(\beta)$ to ensure that we will obtain a simulation, e.g. if $g(\alpha) = g(\beta) = \rightarrow$, we shall want to ensure that $d(\alpha_1, \beta_1) \leqslant \langle 2,1\rangle$, and $d(\alpha_2, \beta_2) \leqslant \langle 1,2\rangle$.

We identify three important concepts in deriving a solution for some set of distance inequalities $\mathcal{I}$

1. A *Q-state* S expresses the minimal distance between some unknown and a set of types in $\mathcal{I}$.

2. A *choice function* g defines the constructor for a particular state.
3. Given a state S and $g(S)$, for each label $l \in L(g(S))$ the *l-successor* state for S at l is obtained as the *$\mathcal{I}$-expansion* of the inequalities derived using the decomposition rules.

4.2.3 Definition [States]: An *state* is a partial function

$$S : \tau \to \{e \in \mathcal{L} \mid e \leqslant \delta_Q\}$$

If P is any set of pairs $\{(\tau, e)\}$, the *Q-reduction* of P is the state

$$\mathrm{red}(P)(\tau) = \delta_Q \wedge \bigwedge\{e' \mid (\tau, e') \in P\} \qquad \text{if } (\tau, e') \in P \text{ for some } e'$$

Similarly the Q-reduction of a state S is the state

$$\mathrm{red}(S)(\tau) = \delta_Q \wedge S(\tau) \qquad \text{if } S(\tau)\downarrow$$

The *binary closure* $B(S)$ of a state S is the set of inequalities

$$\{d(\tau_1, \tau_2) \leqslant S(\tau_1)^{-1} + S(\tau_2) \mid \tau_1, \tau_2 \in \mathrm{dom}(S)\}$$

S is *Q-consistent* if $B(S)$ is Q-consistent.
We define a relation $\lesssim$ on states: $S_1 \lesssim S_2$ if $\mathrm{dom}(S_1) = \mathrm{dom}(S_2)$, and

- $S_1(\tau) \leqslant \delta_Q \wedge (\langle 1, 2\rangle + S_2(\tau))$
- $S_2(\tau) \leqslant \delta_Q \wedge (\langle 2, 1\rangle + S_1(\tau))$

4.2.4 Example: Returning to our earlier example, if S_α and S_β are the initial states for α and β, we have

$$\begin{array}{lcl@{\qquad\qquad}lcl}
S_\alpha(\gamma \to \mathtt{Nat}) & = & \langle 1, 2\rangle & S_\beta(\gamma \to \mathtt{Nat}) & = & \langle 3, 2\rangle \\
S_\alpha(\alpha) & = & \langle 1, 1\rangle & S_\beta(\alpha) & = & \langle 2, 1\rangle \\
S_\alpha(\beta) & = & \langle 1, 2\rangle & S_\beta(\beta) & = & \langle 1, 1\rangle \\
S_\alpha(\mathtt{Long} \to \mathtt{Long}) & = & \langle 2, 3\rangle & S_\beta(\mathtt{Long} \to \mathtt{Long}) & = & \langle 2, 1\rangle
\end{array}$$

and from the definition we can see that $S_\alpha \lesssim S_\beta$. Indeed we also have $S_\alpha \lesssim S_{\gamma \to \mathtt{Long}}$, and $S_{\mathtt{Nat} \to \mathtt{Int}} \lesssim S_\beta$.

4.2.5 Remark: Suppose Q is a Helly poset of base types with HSF $\mathcal{F}_Q$. Our definition of Q-consistency here coincides with that of definition(2.4.4). Moreover if S and T are Q-consistent states defined only on base types in Q and variables and $S \lesssim T$, then $d(S|_Q, T|_Q) \leqslant \langle 1, 2\rangle$, and so $\mathcal{F}_Q(S_1|_Q) \leqslant \mathcal{F}_Q(S_2|_Q)$.

4.2.6 Definition [$\mathcal{I}$-expansion]: Let $\mathcal{I}$ be a set of distance inequalities. The $\mathcal{I}$-expansion of a state S is

$$\begin{array}{rcll} \exp_{\mathcal{I}}(S)(\tau) & = & \delta_Q \wedge \bigwedge\{S(\tau') + d_{\mathcal{I}}(\tau',\tau)\} & \text{if there is } \tau' \text{ such that } S(\tau')\downarrow \text{ and} \\ & & & \quad d_{\mathcal{I}}(\tau',\tau) < e \text{ for some } e \\ & & \text{undefined} & \text{otherwise} \end{array}$$

S is a $\mathcal{I}$*-state* if for every $\tau_1, \tau_2 \in \mathrm{dom}(S)$, $d_{\mathcal{I}}(\tau_1,\tau_2) \leqslant d_{B(S)}(\tau_1,\tau_2)$, so if $\mathcal{I}$ is Q-consistent, then so is any $\mathcal{I}$-state.

4.2.7 Proposition [Properties of $\mathcal{I}$-expansion]: Suppose Q is a partial order of diameter δ_Q. Let $\mathcal{I}$ be a Q-closed set of distance inequalities.

1. if S is an $\mathcal{I}$-state, then so is $\exp_{\mathcal{I}}(S)$.
2. if $S_1 \lesssim S_2$, then $\exp_{\mathcal{I}}(S_1) \lesssim \exp_{\mathcal{I}}(S_2)$

Proof:

1. Let $T = \mathrm{red}(\exp_{\mathcal{I}}(S))$, and suppose $T(\tau_1) = e_1$, $T(\tau_2) = e_2$, so that $d_{B(T)}(\tau_1,\tau_2) = e_1^{-1} + e_2$.

   $$\begin{array}{rl} e_1^{-1} + e_2 = & (\delta_Q \wedge \bigwedge\{S(\tau_1') + d_{\mathcal{I}}(\tau_1',\tau_1) \mid S(\tau_1')\downarrow\})^{-1} + \\ & (\delta_Q \wedge \bigwedge\{S(\tau_2') + d_{\mathcal{I}}(\tau_2',\tau_2) \mid S(\tau_2')\downarrow\}) \end{array}$$

   which is at least

   $$\delta_Q \wedge \bigwedge\{d_{\mathcal{I}}(\tau_1,\tau_1') + d_{B(S)}(\tau_1',\tau_2') + d_{\mathcal{I}}(\tau_2',\tau_2) \mid S(\tau_1')\downarrow, S(\tau_2')\downarrow\}$$

   Now since S is a $\mathcal{I}$-state, we have $d_{\mathcal{I}}(\tau_1',\tau_2') \leqslant d_{B(S)}(\tau_1',\tau_2')$, so

   $$d_{\mathcal{I}}(\tau_1,\tau_2) \leqslant d_{\mathcal{I}}(\tau_1,\tau_1') + d_{B(S)}(\tau_1',\tau_2') + d_{\mathcal{I}}(\tau_2',\tau_2)$$

   for any τ_1' and τ_2', so

   $$d_{\mathcal{I}}(\tau_1,\tau_2) \leqslant \bigwedge\{d_{\mathcal{I}}(\tau_1,\tau_1') + d_{B(S)}(\tau_1',\tau_2') + d_{\mathcal{I}}(\tau_2',\tau_2) \mid S(\tau_1')\downarrow, S(\tau_2')\downarrow\}$$

   And so, since $\mathcal{I}$ is closed under TRUNCATE, we have

   $$d_{\mathcal{I}}(\tau_1,\tau_2) \leqslant \delta_Q \wedge \bigwedge\{d_{\mathcal{I}}(\tau_1,\tau_1') + d_{B(S)}(\tau_1',\tau_2') + d_{\mathcal{I}}(\tau_2',\tau_2) \mid S(\tau_1')\downarrow, S(\tau_2')\downarrow\}$$

2. Suppose $S_1 \lesssim S_2$. Let $T_1 = \exp_{\mathcal{I}}(S_1)$ and $T_2 = \exp_{\mathcal{I}}(S_2)$. If $T_2(\tau_2) = e_2$. Then we must have

   $$e_2 = \delta_Q \wedge \bigwedge\{e_2' + e \mid S_2(\tau) = e_2' \text{ and } d_{\mathcal{I}}(\tau,\tau_2) \leqslant e\}$$

But for any τ such that $S_2(\tau) = e'_2$, we have $S_1(\tau) \leqslant \delta_Q \wedge (\langle 1,2\rangle + e'_2)$. Thus $\tau \in \mathrm{dom}(T_1)$, and moreover

$$\begin{array}{rcl}
T_1(\tau_1) & = & \delta_Q \wedge \bigwedge\{e'_1 + e \mid S_1(\tau) = e'_1 \text{ and } d_{\mathcal{I}}(\tau,\tau_1) \leqslant e\} \\
 & \leqslant & \delta_Q \wedge \bigwedge\{\langle 1,2\rangle + e'_2 + e \mid S_2(\tau) = e'_2 \text{ and } d_{\mathcal{I}}(\tau,\tau_1) \leqslant e\} \\
 & = & \delta_Q \wedge (\langle 1,2\rangle + \bigwedge\{e'_2 + e \mid S_2(\tau) = e'_2 \text{ and } d_{\mathcal{I}}(\tau,\tau_1) \leqslant e\}) \\
 & = & \delta_Q \wedge (\langle 1,2\rangle + e_2)
\end{array}$$

The reasoning for the other condition is dual. □

4.2.8 Definition [Descent Functions]: Suppose Q is a decomposable ordering. Let c be a constructor, τ and $e \in \mathcal{L}$ be such that such that $d_Q(c, \nu(\tau)) \leqslant e$. Let $\rho = c\alpha_{l_1} \ldots \alpha_{l_n}$ We define the *descent function* for Q as follows:

$$\mathcal{P}^l_c(\tau, e) = (\tau_l, e') \ \text{ if } (\rho', \tau_l, e') \in D(\rho, \tau, e) \text{ for some } \rho'$$

We extend this to states: if c is a constructor such that $d(c, \nu(\tau) \leqslant e$ for every constructor type τ such that $S(\tau) = e$

$$\mathcal{P}^l_c(S)(\tau) = \mathrm{red}\{\mathcal{P}^l_c(\tau, e) \mid S(\tau) = e \text{ and } \mathcal{P}^l_c(\tau, e)\downarrow\}$$

4.2.9 Example: If we take S_α and S_β as before, it is clear that we have to choose $\rightarrow$ as the constructor for each state.

$$\begin{array}{lcllcl}
\mathcal{P}^1_{\rightarrow}(S_\alpha)(\gamma) & = & \langle 2,1\rangle & \mathcal{P}^1_{\rightarrow}(S_\alpha)(\texttt{Long}) & = & \langle 3,2\rangle \\
\mathcal{P}^1_{\rightarrow}(S_\beta)(\gamma) & = & \langle 2,3\rangle & \mathcal{P}^1_{\rightarrow}(S_\beta)(\texttt{Long}) & = & \langle 1,2\rangle \\
\mathcal{P}^2_{\rightarrow}(S_\alpha)(\texttt{Nat}) & = & \langle 1,2\rangle & \mathcal{P}^2_{\rightarrow}(S_\alpha)(\texttt{Long}) & = & \langle 2,3\rangle \\
\mathcal{P}^2_{\rightarrow}(S_\beta)(\texttt{Nat}) & = & \langle 3,2\rangle & \mathcal{P}^2_{\rightarrow}(S_\beta)(\texttt{Long}) & = & \langle 2,1\rangle
\end{array}$$

Notice that $\mathcal{P}^1_{\rightarrow}(S_\beta) \lesssim \mathcal{P}^1_{\rightarrow}(S_\alpha)$, and $\mathcal{P}^2_{\rightarrow}(S_\alpha) \lesssim \mathcal{P}^2_{\rightarrow}(S_\beta)$

4.2.10 Definition [Choice Function]: Let S be a state, g be a total function from Q-consistent states to constructors. If $\mathcal{I}$ is a set of distance inequalities, $l \in L(g(S))$ we define the l-successor of S

$$K^l_{\mathcal{I}}(S) = \exp_{\mathcal{I}}(\mathcal{P}^l_{g(S)}(S))$$

Then g is a *choice function* for Q if for any Q-closed $\mathcal{I}$ and any Q-consistent $\mathcal{I}$-state S

G1 $d_Q(g(S), \nu(\tau)) \leqslant e$ for every constructor type τ such that $S(\tau) = e$ (so $K^l_{\mathcal{I}}(S)$ exists).

G2 for any $l \in L(g(S))$, the successor state $K^l_{\mathcal{I}}(S)$ is an $\mathcal{I}$-state.

and for any pair S_1, S_2 of consistent $\mathcal{I}$-states such that $S_1 \lesssim S_2$,

G3 $g(S_1) \sqsubseteq g(S_2)$

G4 For each $l \in L(g(S_1)) \cap L(g(S_2)))$

- if $p^l_{g(S_1)} = \texttt{pos}$, $K^l_{\mathcal{I}}(S_1) \lesssim K^l_{\mathcal{I}}(S_2)$
- if $p^l_{g(S_1)} = \texttt{neg}$, $K^l_{\mathcal{I}}(S_2) \lesssim K^l_{\mathcal{I}}(S_1)$
- if $p^l_{g(S_1)} = \texttt{mix}$, $K^l_{\mathcal{I}}(S_2) \lesssim K^l_{\mathcal{I}}(S_1)$ and $K^l_{\mathcal{I}}(S_1) \lesssim K^l_{\mathcal{I}}(S_2)$

(note that by **G3** and remark (1.3.1), $p^l_{g(S_1)} = p^l_{g(S_2)}$)

4.2.11 Theorem: Let Q be a decomposable ordering and g a choice function for Q. Suppose C is a constraint set whose Q-closure $\mathcal{I}$ is Q-consistent. Then C is solvable.

Proof: For each τ occurring in $\mathcal{I}$ we define the *start state* for τ to be the $\mathcal{I}$-state

$$\text{start}_\tau = \exp_{\mathcal{I}}(\{\tau, \langle 1, 1\rangle\})$$

Let S_C, the set of C-states be the smallest set such that

- $\text{start}_\tau \in S_C$ for every $\tau \in \mathcal{O}(C)$
- $K^l_{\mathcal{I}}(S) \in S_C$ for every $S \in S_C$.

By induction using property **G2**, every state in S_C is a $\mathcal{I}$-state, and since $\mathcal{I}$ is Q-consistent, every state in S_C is Q-consistent.
We define the *state graph* H_C to be the transition graph whose vertex set is S_C and whose transition function is:

$$S \xrightarrow{l} K^l_{\mathcal{I}}(S)$$

Then we define $\nu(S) = g(S)$, and for each $\tau \in \mathcal{O}(C)$ we take $\theta(\tau) = (H_C, \text{start}_\tau)$. Since S_C is finite, $\theta(\tau)$ is a regular type.

To show that θ is a substitution, it suffices to demonstrate that for any constructor type τ, $g(\text{start}_\tau) = \nu(\tau)$ and that for any $l \in L(\tau)$

$$\text{start}_\tau \xrightarrow{l} \text{start}_{\tau(l)}$$

The first follows immediately from **G1**, since $\text{start}_\tau(\tau) = \langle 1, 1\rangle$, For the second, suppose $S = K^l_{\mathcal{I}}(\text{start}_\tau)$. Then we will show that $S = \text{start}_{\tau(l)}$. $S(\tau(l)) = \langle 1, 1\rangle$, so by $\mathcal{I}$-expansion, for any τ' such hat $d_{\mathcal{I}}(\tau(l), \tau') \leqslant e$ for some e,

$$\begin{array}{rcl} S(\tau') & \leqslant & d_{\mathcal{I}}(\tau(l), \tau') \\ & = & \text{start}_{\tau(l)}(\tau') \end{array}$$

but since by **G2** S is an $\mathcal{I}$-state, we have

$$\begin{array}{rcl} \mathrm{start}_{\tau(l)}(\tau') & = & d_{\mathcal{I}}(\tau(l), \tau') \\ & \leqslant & d_{B(S)}(\tau(l), \tau') \\ & = & \langle 1, 1\rangle + S(\tau') = S(\tau') \end{array}$$

Thus $S = \mathrm{start}_{\tau(l)}$.

To see that θ is a solution, observe that if $\tau_1 \leqslant \tau_2 \in C$ then $\mathrm{start}_{\tau_1} \lesssim \mathrm{start}_{\tau_2}$, and by a simple induction using **G3** and **G4**, for any $\tau_1 \leqslant \tau_2 \in C$, there is a simulation $\theta(\tau_1) \lesssim \theta(\tau_2)$, and thus $\theta(\tau_1) \leqslant \theta(\tau_2)$ □

4.3 Examples

Atomic Subtyping and →

The case of structural constructors over some base poset Q_0 is easy. We will work with just →, although others are equally straightforward. Notice that the diameter δ_Q of Q is $\delta_{Q_0} \wedge \delta^*_{Q_0}$. Our descent function is

$$\begin{array}{rcl} \mathcal{P}^1_{\rightarrow}(\tau_1 \rightarrow \tau_2, e) & = & (\tau_1, e^*) \\ \mathcal{P}^2_{\rightarrow}(\tau_1 \rightarrow \tau_2, e) & = & (\tau_2, e) \end{array}$$

As a choice function, we take

$$\begin{array}{rcll} g(S) & = & \rightarrow & \text{if } S(\tau_1 \rightarrow \tau_2)\downarrow \text{ for some } \tau_1 \rightarrow \tau_2 \\ & & F_Q(S|_Q) & \text{if } S(\mathtt{a})\downarrow \text{ for some } \mathtt{a} \in Q \\ & & \mathtt{c} & \text{otherwise} \end{array}$$

For consistent S, $\mathrm{dom}(S)$ cannot contain both an atom and a constructor type, and can contain at most one component of Q, so this is well-defined. It remains to verify that it satisfies conditions **G1**–**G4**.

G1 immediate using remark(4.2.5) for the atomic case

G2 We will prove that $\mathcal{P}^l_{\rightarrow}(S)$ is a $\mathcal{I}$-state for $l = 1, 2$, for then by proposition(4.2.7) $K^l_{\mathcal{I}}(S) = \exp_{\mathcal{I}}(\mathcal{P}^l_{\rightarrow}(S))$ is also. We will do the $l = 1$ case, $l = 2$ is similar. We need to show that for any τ_1, τ_2 such that $\mathcal{P}^1_{\rightarrow}(S)(\tau_1) = e$, $\mathcal{P}^1_{\rightarrow}(S)(\tau_2) = e'$, $d_{\mathcal{I}}(\tau_1 \tau_2) \leqslant e^{-1} + e'$. It suffices to demonstrate that for every τ_1', τ_2' such that $S(\tau_1 \rightarrow \tau_1') = e_1$ and $S(\tau_2 \rightarrow \tau_2') = e_2$, that $d_{\mathcal{I}}(\tau_1,\tau_2) \leqslant \delta_Q \wedge ((e_1^*)^{-1} + e_2^*)$, for then

$$\begin{array}{rcl}
d_{\mathcal{I}}(\tau_1,\tau_2) & \leqslant & \bigwedge_{\tau_1',\tau_2'}\{\delta_Q \wedge ((e_1^*)^{-1} + e_2^*) \mid S(\tau_1{\rightarrow}\tau_1') = e_1, S(\tau_2{\rightarrow}\tau_2') = e_2\} \\
& = & \delta_Q \wedge \left(\bigwedge_{\tau_1',\tau_2'}\{(e_1^*)^{-1} + e_2^* \mid S(\tau_1{\rightarrow}\tau_1') = e_1, S(\tau_2{\rightarrow}\tau_2') = e_2\}\right) \\
& = & \delta_Q \wedge \left(\bigwedge_{\tau_1'}(S(\tau_1{\rightarrow}\tau_1')^*)^{-1} + \bigwedge_{\tau_2'} S(\tau_2{\rightarrow}\tau_2')^*\right) \\
& & \quad \text{by distributivity of } + \text{ over } \wedge \\
& = & \left(\bigwedge_{\tau_1'} S(\tau_1{\rightarrow}\tau_1')^{-1}\right)^* + \left(\bigwedge_{\tau_2'} S(\tau_2{\rightarrow}\tau_2')\right)^* \\
& = & \left(\bigwedge_{\tau_1'} S(\tau_1{\rightarrow}\tau_1')^*\right)^{-1} + \left(\bigwedge_{\tau_2'} S(\tau_2{\rightarrow}\tau_2')\right)^* \\
& = & (\mathcal{P}^1_{\rightarrow}(S)(\tau_1))^{-1} + \mathcal{P}^1_{\rightarrow}(S)(\tau_2)
\end{array}$$

But if $S_1(\tau_1{\rightarrow}\tau_1') = e_1$ and $S_2(\tau_2{\rightarrow}\tau_2') = e_2$, then $d_{B(S)}(\tau_1{\rightarrow}\tau_1', \tau_2{\rightarrow}\tau_2') \leqslant e_1^{-1}{+}e_2$, so since S is a $\mathcal{I}$-state, $d_{\mathcal{I}}(\tau_1{\rightarrow}\tau_1', \tau_2{\rightarrow}\tau_2') \leqslant e_1^{-1}{+}e_2$, so by Decomp, $d_{\mathcal{I}}(\tau_1,\tau_2) \leqslant (e_1^{-1} + e_2)^* = (e_1^*)^{-1} + e_2^*$.

G3 Suppose $S \lesssim T$. Then $\mathrm{dom}(S) = \mathrm{dom}(T)$. If the set contains →-types, or only variables, the result is immediate, and if $\mathrm{dom}(S)$ and $\mathrm{dom}(T)$ both contain base types, by remark(4.2.5) we have $F_Q(S|_Q) \leqslant F_Q(T|_Q)$.

G4 Suppose $S \lesssim T$, we will take $l = 1$, although the $l = 2$ case is similar. We define $S_1 = \mathcal{P}^1_{\rightarrow}(S)$, $T_1 = \mathcal{P}^1_{\rightarrow}(T)$. By proposition (4.2.7), it suffices to show that $T_1 \lesssim S_1$. Suppose $T_1(\tau) = e$, then we must have

$$\begin{array}{rcl}
S_1(\tau) & = & \delta_Q \wedge \left(\bigwedge_{\tau'} S(\tau{\rightarrow}\tau')\right)^* \\
& & \quad \text{over all } \tau' \text{ such that } S(\tau{\rightarrow}\tau')\downarrow \\
& \leqslant & \delta_Q \wedge \left(\bigwedge_{\tau'} \delta_Q \wedge (\langle 1,2\rangle + T(\tau{\rightarrow}\tau'))\right)^* \\
& \leqslant & \delta_Q \wedge \left(\langle 2,1\rangle + \left(\bigwedge_{\tau'} T(\tau{\rightarrow}\tau')\right)^*\right) \\
& \leqslant & \delta_Q \wedge (\langle 2,1\rangle + T_1(\tau))
\end{array}$$

Similarly $T_1(\tau) \leqslant \delta_Q \wedge (\langle 1,2\rangle + S_1(\tau))$, so $T_1 \lesssim S_1$

Atomic Subtyping, →, and {}

The argument for records is very similar. Our descent function is

$$\begin{array}{rcll}
\mathcal{P}^1_{\rightarrow}(\tau_1{\rightarrow}\tau_2, e) & = & (\tau_1, e^*) & \\
\mathcal{P}^2_{\rightarrow}(\tau_1{\rightarrow}\tau_2, e) & = & (\tau_2, e) & \\
\mathcal{P}^l_{\{\}_I}(\{l_j{:}\tau_j\}, e) & = & (\tau_l, e) & e < \langle 2,2\rangle \text{ and } l \in I \cap J \\
\mathcal{P}^l_{\{\}_I}(\{l_j{:}\tau_j\}, e) & = & (\tau_l, \langle 3,2\rangle) & \langle 2,2\rangle \leqslant e \leqslant \langle 3,2\rangle \text{ and } l \in I \cap J
\end{array}$$

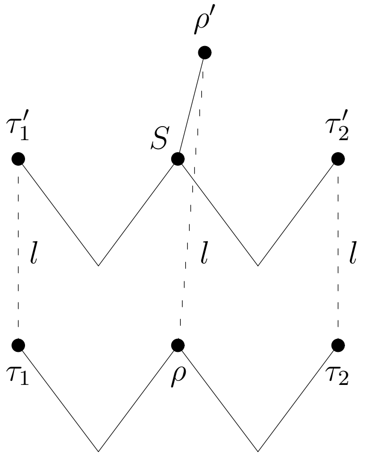
and our choice function

$$\begin{array}{rcll} g(S) & = & \rightarrow & \text{if } S(\tau_1{\rightarrow}\tau_2){\downarrow} \text{ for some } \tau_1{\rightarrow}\tau_2 \\ & & F_Q(S|_Q) & \text{if } S(\mathsf{a}){\downarrow} \text{ for some } \mathsf{a} \in Q \\ & & \{\}_{L(S)} & \text{if } S(\{l_i{:}\tau_i\}){\downarrow} \text{ for some } \{l_i{:}\tau_i\} \\ & & & \qquad \text{where } L(S) = \bigcup\{L(\tau) \mid S(\tau) \leqslant \langle 1,2\rangle\} \\ & & \mathsf{c} & \text{otherwise} \end{array}$$

Again notice that exactly one of the above cases can hold by consistency of S.

G1 immediate

G2 The → case follows as in example(4.3) So suppose $g(S) = \{\}_L$. Again it suffices to show that $\mathcal{P}^l_{\{\}_L}(S)$ is an $\mathcal{I}$-state for each $l \in L$, then the result follows by proposition(4.2.7). So suppose $S_l = \mathcal{P}^l_{\{\}_L}$, and $S_l(\tau_1) = e_1$, $S_l(\tau_2) = e_2$. There are many possible cases to consider, where each of e_1, e_2 is $\langle 1,1\rangle$, $\langle 1,2\rangle$, $\langle 2,1\rangle$, $\langle 3,2\rangle$. We will do the case where $e_1 = e_2 = \langle 3,2\rangle$, the others are similar.



Since $l \in L(g(S))$, there must be ρ' such that $S(\rho') \leqslant \langle 1,2\rangle$, and $l \in L(\rho')$. By inspection of the descent function, we must have τ_1' and τ_2' such that $\tau_1'(l) = \tau_1$, $\tau_2'(l) = \tau_2$, and $\langle 2,2\rangle \leqslant S(\tau_1') \leqslant \langle 3,2\rangle$, and $\langle 2,2\rangle \leqslant S(\tau_2') \leqslant \langle 3,2\rangle$. If $\rho'(l) = \rho$, we thus have $\{d_{\mathcal{I}}(\tau_1', \rho) \leqslant \langle 3,2\rangle, d_{\mathcal{I}}(\tau_2', \rho) \leqslant \langle 3,2\rangle\} \subseteq B(S)$, so since S is a $\mathcal{I}$-state, $d_{\mathcal{I}}(\tau_1', \rho) \leqslant \langle 3,2\rangle$ and $d_{\mathcal{I}}(\tau_2', \rho) \leqslant \langle 3,2\rangle$. Thus by Decomp, we have $d_{\mathcal{I}}(\tau_1, \rho) \leqslant \langle 3,2\rangle$ and $d_{\mathcal{I}}(\tau_2, \rho) \leqslant \langle 3,2\rangle$, and so by Dual and Join, we have $d_{\mathcal{I}}(\tau_1, \tau_2) \leqslant \langle 5,4\rangle$ as required.

G3 Suppose $S \lesssim T$. If $\mathrm{dom}(S) = \mathrm{dom}(T)$ contains no record types, the result follows in the manner of example (4.3). Otherwise for any $\tau = \{l_i{:}\tau_i\}$ such that $T(\tau) \leqslant \langle 1,2\rangle$, $S(\tau) \leqslant \langle 1,2\rangle$, so $L(S) \supseteq L(T)$, and thus $g(S) \sqsubseteq g(T)$.

G4 The → case is as for example(4.3). Suppose $S \lesssim T$, and $g(S) = \{\}_{L_S}$, and $g(T) = \{\}_{L_T}$, where $L_S \subseteq L_T$. We define $S_l = K^l_{\mathcal{I}}(S)$, $T_l = K^l_{\mathcal{I}}(T)$.

Suppose $T_l(\tau_l) = e$, we need to show that $S_l(\tau_l) \leqslant \langle 1,2\rangle + e$. The cases where $e \leqslant \langle 1,2\rangle$ are as in example(4.3); the interesting cases are $\langle 2,1\rangle \leqslant e \leqslant \langle 3,2\rangle$. As all these cases are essentially similar, we will do the case where $T_l(\tau_l) = \langle 2,1\rangle$, and show that $S_l(\tau_l) \leqslant \langle 2,3\rangle$. There are two possibilities:

- there is some τ such that $\mathcal{P}^l_{\{\}_T}(T)(\tau) = \langle 1,1\rangle$ and $d_{\mathcal{I}}(\tau,\tau_l) = \langle 2,1\rangle$, in which by reasoning similar to example(4.3) we have $\mathcal{P}^l_{\{\}_S}(S)(\tau) \leqslant \langle 1,2\rangle$, and so $S_l(\tau_l) \leqslant \langle 1,2\rangle + \langle 2,1\rangle = \langle 2,3\rangle$.
- there is some τ such that $\mathcal{P}^l_{\{\}_T}(T)(\tau) = \langle 2,1\rangle$ and $d_{\mathcal{I}}(\tau,\tau_l) = \langle 2,1\rangle$. In this case, we must have some τ' such that $T(\tau') = \langle 2,1\rangle$, with $\tau'(l) = \tau$. But since $l \in L(g(T))$, there must also be ρ' such that $T(\rho') = \langle 1,2\rangle$ and $l \in L(\rho)$. In this case, since T is an $\mathcal{I}$-state, we must have

$$\begin{array}{rcl} d_{\mathcal{I}}(\rho',\tau') & \leqslant & d_{B(T)}(\rho',\tau') \\ & = & T(\rho')^{-1} + T(\tau') \\ & = & \langle 2,1\rangle + \langle 2,1\rangle \\ & = & \langle 2,1\rangle \end{array}$$

and thus if $\rho'(l) = \rho$, $d_{\mathcal{I}}(\rho,\tau_l) \leqslant \langle 2,1\rangle$. But since $S(\rho') \leqslant \langle 1,2\rangle$, we have $\mathcal{P}^l_{\{\}_S}(S)(\rho) \leqslant \langle 1,2\rangle$ and thus $S_l(\tau_l) \leqslant \langle 2,3\rangle$ as required.

The reasoning for the other condition is dual.

Atomic Subtyping, →, ⊤ and ⊥

Observe that in this case we have $\delta_Q = \langle 2,2\rangle$ and Q has only one component. Our descent function is

$$\begin{array}{rcll} \mathcal{P}^1_{\rightarrow}(\tau_1{\rightarrow}\tau_2, e) & = & (\tau_1, e^*) & e < \langle 2,2\rangle \\ \mathcal{P}^2_{\rightarrow}(\tau_1{\rightarrow}\tau_2, e) & = & (\tau_2, e) & e < \langle 2,2\rangle \end{array}$$

and our choice function

$$\begin{array}{rcll} g(s) & = & \top & \text{if there is no } S(\mathtt{a}) \leqslant \langle 1,2\rangle \text{ or } S(\tau_1{\rightarrow}\tau_2) \leqslant \langle 1,2\rangle \\ & & \bot & \text{otherwise, if there is no } S(\mathtt{a}) \leqslant \langle 2,1\rangle \text{ or } S(\tau_1{\rightarrow}\tau_2) \leqslant \langle 2,1\rangle \\ & & \rightarrow & \text{otherwise if there is some } S(\tau_1{\rightarrow}\tau_2) < \langle 2,2\rangle \\ & & F_{Q'}(S|_{Q'}) & \text{otherwise, if there is some } S(\mathtt{a}) < \langle 2,2\rangle \end{array}$$

where $Q' = Q \cup \{\top, \bot\}$, which by proposition(2.2.8) is a lattice.

Unlike the previous examples, S is defined on all types occurring in $\mathcal{I}$. However, at most one of the last two cases is possible for any consistent S: if $g(S) \neq \top$ and $g(S) \neq \bot$, then we must have τ_1 and τ_2 such that $S(\tau_1) \leqslant \langle 1,2\rangle$ and $S(\tau_2) \leqslant \langle 2,1\rangle$, and such that each of τ_1 and τ_2 is either an →-type or a base type. It cannot be the case that one is an →-types and one a base type, for then we would have $d(\tau_1, \tau_2) \leqslant \langle 2, 1\rangle \in B(S)$, and $B(S)$ would not be consistent.

G1 immediate

G2 The only case where $L(g(S))$ is non-empty is when $g(S) = \rightarrow$, whence the result follows in a similar manner to example(4.3)

G3 Suppose $S \lesssim T$. if $g(T) = \top$ then the result is immediate. If $g(T) = \bot$, then we must have $T(\tau) = \langle 1, 2\rangle$ for some →-type or atom τ, so $S(\tau) = \langle 1, 2\rangle$, and furthermore there can be no →-type or atom τ' such that $S(\tau') \leqslant \langle 2, 1\rangle$, since then we would have $T(\tau') \leqslant \langle 2, 1\rangle$, contradicting $g(T) = \bot$. Thus $g(S) = \bot$. If $g(T) = \rightarrow$, then we must have some $T(\tau_1{\rightarrow}\tau_2) \leqslant \langle 1, 2\rangle$, and thus $S(\tau_1{\rightarrow}\tau_2) \leqslant \langle 1, 2\rangle$, so $g(S) = \rightarrow$ or $\bot$. Finally, if $g(T) = \mathtt{a}$, by similar reasoning either $g(S) = \bot$, or $g(S) = \mathtt{b}$ where $\mathtt{b} \leqslant \mathtt{a}$ by the fact that $F_{Q'}$ is a HSF.

G4 Suppose $l \in L(g(S)) \cap L(g(T))$. We must have $g(S) = g(T) = \rightarrow$. We will do the $l = 2$ case. Since $K^2_{\mathcal{I}}(S)(\tau) \leqslant \langle 2, 2\rangle$ for every $\tau \in \mathcal{O}(S)$, it suffices to demonstrate that for every τ such that $K^2_{\mathcal{I}}(T) \leqslant \langle 1, 2\rangle$, $K^2_{\mathcal{I}}(S)(\tau) \leqslant \langle 1, 2\rangle$, and for every τ' such that $K^2_{\mathcal{I}}(S)(\tau) \leqslant \langle 2, 1\rangle$, $K^2_{\mathcal{I}}(T)(\tau) \leqslant \langle 2, 1\rangle$. Since these are dual, we will do just the first: suppose we have $K^2_{\mathcal{I}}(T)(\tau) \leqslant \langle 1, 2\rangle$, then we must have τ' such that $\mathcal{P}^l_{\rightarrow}(T)(\tau') \leqslant \langle 1, 2\rangle$, and $d_{\mathcal{I}}(\tau', \tau) \leqslant \langle 1, 2\rangle$. But then we must have $T(\tau''{\rightarrow}\tau') \leqslant \langle 1, 2\rangle$ for some τ'', and thus $S(\tau''{\rightarrow}\tau') \leqslant \langle 1, 2\rangle$. Hence $\mathcal{P}^l_{\rightarrow}(S)(\tau') \leqslant \langle 1, 2\rangle$, and thus $K^2_{\mathcal{I}}(S)(\tau) \leqslant \langle 1, 2\rangle$.

Atomic Subtyping, →, and Nonatomic Base Types

In order to guarantee regular solutions, we require the underlying poset of types to have finite diameter. In general, bounded quantification can result in posets of arbitrarily large diameter, but a finite signature always results in a poset of finite diameter.

4.3.1 Proposition: Suppose Q_Σ is the set of regular types built using the → constructor over a Helly poset Q_0 of finite diameter extended with a bounded signature Σ. Then Q_Σ has finite diameter.

Proof: We proceed by induction over the signature. It is easy to show that if Q_0 has diameter δ_{Q_0}, the poset of regular types over Q has diameter $\delta_{Q_0} \wedge \delta^*_{Q_0}$.

So suppose the poset of regular types over Q extended by Σ has diameter $\delta_{(Q,\Sigma)}$, and $\Sigma' = \Sigma, \mathtt{a} \leqslant \tau$.

For any two types τ_1, τ_2 containing $\mathtt{a}$, let $\tau_1' = [\tau/\mathtt{a}]\tau_1$, and $\tau_2' = [\tau/\mathtt{a}]\tau_2$. Then $d(\tau_1, \tau_1') \leqslant \langle 2, 2\rangle$, since we obtain an up-fence of length 2 from τ to τ' by first promoting all occurrences of $\mathtt{a}$ at positive parity paths, then all occurrences at negative parity , and by doing the reverse, we obtain a down-fence of length 2 from τ_1 to τ_1'; similarly $d(\tau_2, \tau_2') \leqslant \langle 2, 2\rangle$. Now since τ_1' and τ_2' contain no occurrences of $\mathtt{a}$, we must have $d(\tau_1', \tau_2') \leqslant \delta_{(Q,\Sigma)}$, and thus $d(\tau_1', \tau_2') \leqslant \langle 2, 2\rangle + \delta_{(Q,\Sigma)} + \langle 2, 2\rangle$, which is an upper bound for the diameter of $\delta_{(Q,\Sigma')}$. □

Unfortunately, even if Q is a low-diameter poset such as a tree, so that the component of types of shape $\mathtt{a}{\rightarrow}\mathtt{a}$ has diameter $\langle 3, 3\rangle$, adjoining a bounded constant results in a component of diameter $\langle 3, 4\rangle$. Then the component of shape $(\mathtt{a}{\rightarrow}\mathtt{a}){\rightarrow}(\mathtt{a}{\rightarrow}\mathtt{a})$ has diameter $\langle 4, 4\rangle$, and by extending the signature the diameter of the poset can become arbitrarily high.
A base type in Σ is *atomic* if either $\Sigma(\mathtt{a})\uparrow$, or $\Sigma^{\uparrow}(\mathtt{a})$ is an atomic base type (in Q or Σ.)

4.3.2 Proposition: Let Q' be the extension of Q by all the atomic base types in Σ. Q' is a Helly poset.

Proof: By induction on the number of bindings for atomic base types Σ. If $\Sigma(\mathtt{a})\uparrow$, $\{\mathtt{a}\} \cup Q'$ is obviously Helly, otherwise the result follows from proposition(2.2.7). □

Since Q' is Helly, it has a HSF. Our descent function $\mathcal{P}^l_c$ is:

$$\begin{array}{lcll}
\mathcal{P}^1_{\rightarrow}(\tau_1{\rightarrow}\tau_2, e) & = & (\tau_1, e^*) & \\
\mathcal{P}^2_{\rightarrow}(\tau_1{\rightarrow}\tau_2, e) & = & (\tau_2, e) & \\
\mathcal{P}^1_{\rightarrow}(\mathtt{a}, e) & = & (\tau_1, (e^{\blacktriangle})^*) & \text{where } \Sigma^{\Uparrow}(\mathtt{a}) = \tau_1{\rightarrow}\tau_2 \\
\mathcal{P}^2_{\rightarrow}(\mathtt{a}, e) & = & (\tau_2, e^{\blacktriangle}) & \text{where } \Sigma^{\Uparrow}(\mathtt{a}) = \tau_1{\rightarrow}\tau_2 \\
\mathcal{P}^1_{\mathtt{a}}(\tau_1{\rightarrow}\tau_2, e) & = & (\tau_1, (e^{\vartriangle})^*) & \\
\mathcal{P}^2_{\mathtt{a}}(\tau_1{\rightarrow}\tau_2, e) & = & (\tau_1, e^{\vartriangle}) &
\end{array}$$

and our choice function

$$\begin{array}{ll}
\rightarrow & \text{if } S(\tau_1{\rightarrow}\tau_2)\downarrow \text{ for some } \tau_1{\rightarrow}\tau_2 \\
\bigwedge\{\mathtt{a} \mid S(\mathtt{a}) \leqslant \langle 1, 2\rangle\} & \text{if } S(\mathtt{a}) \leqslant \langle 1, 2\rangle \text{ for some non-atomic base type } \mathtt{a} \\
F_{Q'}(S|_{Q'}) & \text{if } \mathrm{dom}(S) \text{ contains an atomic base type} \\
\mathtt{c} & \text{otherwise}
\end{array}$$

Now we need to demonstrate that **G1**–**G4** are met.

G1 straightforward

G2 It suffices to prove that $\mathcal{P}^l_{g(S)}(S)$ is an $\mathcal{I}$-state for then by proposition(4.2.7) so is $K^l_{\mathcal{I}}(S)$. Suppose $g(S) = \rightarrow$, $l = 2$. If $S(\mathtt{a})\uparrow$ for any $\mathtt{a}$, the case is as in example(4.3). So suppose we have $S(\mathtt{a}) = f$ such that $e_1 = f^{\blacktriangle}$, $\Sigma^{\Uparrow}(\mathtt{a}) = \tau_1' \rightarrow \tau_1$, and $S(\tau_2' \rightarrow \tau_2) = e_2$. Then $d_{B(S)}(\tau_1' \rightarrow \tau_1, \tau_2' \rightarrow \tau_2) \leqslant f^{-1} + e_2$, so since S is an $\mathcal{I}$-state, $d_{\mathcal{I}}(\tau_1' \rightarrow \tau_1, \tau_2' \rightarrow \tau_2) \leqslant f^{-1} + e_2$. Then

$$\begin{array}{rcll} d_{\mathcal{I}}(\tau_1, \tau_2) & \leqslant & (f^{-1} + e_2)^{\vartriangle} & \text{by DESCENT} \\ & \leqslant & (f^{-1})^{\vartriangle} + e_2 & \\ & = & (f^{\blacktriangle})^{-1} + e_2 & \\ & = & e_1^{-1} + e_2 & \end{array}$$

The other possibilities are similar.

G3 Suppose $S \lesssim T$.

- If $g(S) = g(T) = \rightarrow$, the result is immediate.
- If $g(S) = \mathtt{a}$ and $g(T) = \mathtt{b}$, there are two cases: either $\mathtt{a}, \mathtt{b} \in Q'$, whence by remark(4.2.5) we have $F_{Q'}(S|_{Q'}) \leqslant F_{Q'}(T|_{Q'})$. Or $\mathtt{a}$ and $\mathtt{b}$ are nonatomic base types, in which case for every $\mathtt{c}$ such that $T(\mathtt{c}) \leqslant \langle 1, 2 \rangle$, we must have $S(\mathtt{c}) \leqslant \langle 1, 2 \rangle$, so by the definition of g, $\mathtt{a} \leqslant \mathtt{b}$.
- If $g(T) = \mathtt{a}$, we must have $T(\mathtt{a}) \leqslant \langle 1, 2 \rangle$, so $S(\mathtt{a}) \leqslant \langle 1, 2 \rangle$, and so it cannot be the case that $g(S) = \rightarrow$.
- If $g(S) = \mathtt{a}$ and $g(T) = \rightarrow$, by consistency $\Sigma^{\Uparrow}(\mathtt{a})$ must be an arrow type and so $\mathtt{a} \sqsubseteq \rightarrow$

G4 Suppose $S \lesssim T$ and $g(S) = g(T) = \rightarrow$, again we will do the $l = 2$ case. If we have $T_2(\tau) = e$, and $T(\tau' \rightarrow \tau) = e$, then the result follows as in example(4.3). Otherwise suppose we have $T(\mathtt{a}) = f$, with $\Sigma^{\Uparrow}(\mathtt{a}) = \tau' \rightarrow \tau$, and $f^{\blacktriangle} = e$. Then we have $S(\mathtt{a}) \leqslant \delta_Q \wedge (\langle 1, 2 \rangle + f)$, and thus $S_2(\tau) \leqslant \delta_Q \wedge (\langle 1, 2 \rangle + f)^{\blacktriangle}$, so $S_2(\tau) \leqslant \delta_Q \wedge (\langle 1, 2 \rangle + f^{\blacktriangle})$. If $g(S)$ or $g(T)$ is a base type, the argument is similar.

If $g(S) = \mathtt{a}$ and $g(T) = \mathtt{b}$ then $\mathtt{a} \leqslant \mathtt{b}$ and so for any τ such that $S_2(\tau) = e$ and $e = f^{\blacktriangle}$, for any $S(\tau' \rightarrow \tau) = f$ we have $T(\tau' \rightarrow \tau) \leqslant \langle 2, 1 \rangle + f$, and thus

$$\begin{array}{rcl} T_2(\tau) & \leqslant & \delta_Q \wedge (\langle 2, 1 \rangle + f)^{\blacktriangle} \\ & = & \delta_Q \wedge (\langle 2, 1 \rangle + f^{\blacktriangle}) \\ & = & \delta_Q \wedge (\langle 2, 1 \rangle + e) \end{array}$$

and similarly $S_2(\tau) \leqslant \delta_Q \wedge (\langle 1, 2 \rangle + T_2(\tau))$ so $S_2 \lesssim T_2$.

One application of this result is to rephrase an entailment problem as a solvability problem. We introduce *bounded* quantifiers, which quantify only over a single variable and a single constraint which bounds that variable above; and *unbounded* quantifiers, which quantify only over a single variable without constraint.

4.3.3 Definition [Types and Type Schemes]:

$$\begin{array}{lll}
\text{Types} & \tau & ::= \ \mathtt{a} \ \mid \ \alpha \ \mid \ \mathtt{T}\tau_1 \dots \tau_n \\
\text{Bounded Type Schemes} & \sigma & ::= \ \tau \ \mid \ \forall\alpha.\sigma \ \mid \ \forall\alpha{\leqslant}\tau.\sigma \\
\text{Constrained Type Schemes} & \kappa & ::= \ \forall\overline{\alpha}\backslash C.\tau
\end{array}$$

where C is a set of inequalities of the form $\tau \leqslant \tau'$.

Bounded type schemes are relatively easy to read, and thus provide a programmer-friendly way to print out those constrained types which may be rendered in such a form.

4.3.4 Definition [Instances and Entailment]: We say that τ is a (Q, Σ)-*instance* of $\forall\overline{\alpha}\backslash C.\tau'$ if there is a substitution $\theta : \overline{\alpha} {\to} \tau(Q \cup \text{atom}(\Sigma))$ such that $Q, \Sigma \ \vdash \ \theta c$ for each $c \in C$, and $\theta\tau' = \tau$.

We define $Q, \Sigma \ \vdash \ \kappa \preccurlyeq \sigma$ if for any extension Σ' of Σ and Q-model Q', for any (Q', Σ')-instance τ_σ of σ, there is a (Q', Σ')-instance τ_κ of κ such that $\Sigma' \ \vdash \ \tau_\kappa \leqslant \tau_\sigma$

Like other entailment relations in the literature, and like the semantic notion of entailment of chapter 3, this definition is based on the existence of an instance of τ_κ which is a subtype of any notion of τ_σ under any possible extension of the underlying order. The relation given here is much weaker than that of [52] or [40], where the underlying order may be extended by arbitrary constraints, or even recursive constraints in which a type variable is allowed to occur on both sides of an inequality. However, we gain the ability to test the entailment relation efficiently by checking solvability of an appropriate constraint set.

4.3.5 Proposition: Suppose $\sigma = \forall\alpha_1({\leqslant}\tau_1)\forall\alpha_2({\leqslant}\tau_2) \dots \forall\alpha_n({\leqslant}\tau_n).\tau_\sigma$ and $\kappa = \forall\overline{\beta}\backslash C.\tau_\kappa$. Then $Q, \Sigma \ \vdash \ \kappa \preccurlyeq \sigma$ iff for

$$\Sigma' = \Sigma, \mathtt{a}_1({\leqslant}\tau_1), \mathtt{a}_2({\leqslant}[\mathtt{a}_1/\alpha_1]\tau_2) \dots , \mathtt{a}_n({\leqslant}[\mathtt{a}_1/\alpha_1, \dots , \mathtt{a}_{n-1}/\alpha_{n-1}]\tau_n)$$

there is some substitution θ from $\overline{\beta}$ to $\tau(Q \cup \Sigma')$ such that $Q, \Sigma' \ \vdash \ \theta(c)$ for every $c \in C \cup \{\theta\tau_\kappa \leqslant [\mathtt{a}_i/\alpha_i]\tau_\sigma\}$.

Proof: Suppose $Q, \Sigma \ \vdash \ \kappa \leqslant \sigma$. Then by the definition of the relation $\leqslant$, we know that for any (Q, Σ')-instance of σ, there is a (Q, Σ')-instance of κ which is a subtype of it. Now $[\mathtt{a}_i/\alpha_i]\tau_\sigma$ is an instance of σ, so we must have some substitution θ on the variables $\overline{\beta}$ quantified over in κ, such that $Q, \Sigma' \vdash \theta(c)$ for every $c \in C$, and moreover $Q, \Sigma' \vdash \theta\tau_\kappa \leqslant [\mathsf{a}_i/\alpha_i]\tau_\sigma$ as required.

Now suppose we have θ such that $Q, \Sigma' \vdash \theta(c)$ for every $c \in C \cup \{\theta\tau_\kappa \leqslant [\mathsf{a}_i/\alpha_i]\tau_\sigma\}$, and suppose Σ_1 is an extension of Σ and $Q_1 \rhd Q$. Without loss of generality we may assume that the new base types in Σ_1 and Q_1 are distinct from the a_i. Let θ_1 be a substitution from the α_i to $\tau(Q_1 \cup \mathrm{atom}(\Sigma_1))$ such that $Q_1, \Sigma_1 \vdash \theta_1\alpha_i \leqslant \theta_1\tau_i$, so that $\theta_1\tau_\sigma$ is a (Q_1, Σ_1)-instance of σ. Let $\theta' = \theta_1 \circ [\alpha/\mathsf{a}] \circ \theta$. We claim that $Q_1, \Sigma_1 \vdash \theta'\tau_\kappa \leqslant \theta_1\tau_\sigma$, and that $Q_1, \Sigma_1 \vdash \theta' C$.

These follow from the fact that if $Q, \Sigma' \vdash \tau_1 \leqslant \tau_2$, then $Q, \Sigma_1 \vdash (\theta_1 \circ [\alpha_i/\mathsf{a}_i])\tau_1 \leqslant (\theta_1 \circ [\alpha_i/\mathsf{a}_i])\tau_2$, which we will show by induction on the structure of the proof of $Q, \Sigma' \vdash \tau_1 \leqslant \tau_2$.

Suppose the result holds for the premises of a rule. If the rule is Refl or QTax, the result is immediate, and if it is Cons, then the result is follows from the induction hypothesis. The remaining case is Prom$^\uparrow$: If the variable being promoted is not one of the a_i, then the result follows from the induction hypothesis. Otherwise, the conclusion of the rule is $Q, \Sigma' \vdash \mathsf{a}_m \leqslant \tau_2$, and it has as a premise $Q, \Sigma' \vdash \Sigma(\mathsf{a}_m) \leqslant \tau_2$. So by the induction hypothesis, we have $Q_1, \Sigma_1 \vdash (\theta_1 \circ [\alpha_i/\mathsf{a}_i])(\Sigma'^\uparrow(\mathsf{a}_m)) \leqslant (\theta_1 \circ [\alpha_i/\mathsf{a}_i])\tau_2$. But $(\theta_1 \circ [\alpha_i/\mathsf{a}_i])(\Sigma'^\uparrow(\mathsf{a}_m))$ is just $\theta_1\tau_m$, and we have $Q_1, \Sigma_1 \vdash \theta_1\alpha_i \leqslant \theta_1\tau_i$, so by transitivity we obtain the required result $Q_1, \Sigma_1 \vdash (\theta_1 \circ [\alpha_i/\mathsf{a}_i])(\mathsf{a}_m) \leqslant (\theta_1 \circ [\alpha_i/\mathsf{a}_i])\tau_2$ □

This presents us with an alternative formulation of the subtyping relation which is not parameterised over all possible extensions of Σ, and which will be the basis for checking the relation: We simply extend the signature as relevant and check for solvability.

4.4 Conclusion

We have demonstrated a single framework which encompasses both atomic subtyping over Helly posets, and the various possibilities for structural and nonstructural subtyping. In particular, our techniques allow us to deal with the expressions given in the introduction, illustrating the problems that can result from the inclusion of $\top$ and $\bot$ in a type system. We now have the ability to check the subtype relation without them, but retaining records. In the context of record and function types only, we return to the term:

```
let f = fun(r) r.X + if r.X then 1 else 2;
```

The "One big lattice" technique accepts this term as well-typed: in fact it has the minimal type $\{X{:}\bot\}{\rightarrow}\texttt{Nat}$, since the `X` field of `r` is used at types `Bool` and `Int`. However, using the results of this chapter we have an effective technique which enables us to denounce this term as containing a type error. Similarly, in the presence of polymorphic equality, we reject the term `true = 3`.

Our main theorem for solvability are also applicable to other type systems

- AC-object constructors [2] whose inference problem is studied in [32, 20],
- systems with $\top$ and not $\bot$ (or conversely), introduced in [48], whose inference problem is studied in [24, 33]
- Systems of records with covariant and invariant field subtyping [44]

Our main extension to the studies of type inference in these systems is the incorporation of atomic subtyping, and the development of a unifying framework in which to consider combinations of the various distinctive features of such constructors.

Polynomial Time Solution

It is not had to see that for the orderings we have considered, the Q-closure of a constraint set can be accomplished by dynamic transitive distance closure in polynomial time, thus providing a test for solvability. However, in [49], Tiuryn shows that even when solutions are represented as directed acyclic graphs, the size of a minimal solution grows exponentially with respect to the size of the constraint set, and so we cannot expect to construct solutions in polynomial time. For finite structural subtyping over a lattice, Tiuryn gives a method for constructing an *augmented DAG* for each variables in a solvable constraint set: a graph of polynomial size from which the solution at any particular node in the solution tree can be recovered in polynomial time.

Instead of constructing distinct graphs for each variable, we can use a single graph (the *Helly Closure Graph* for the constraint set C) whose vertices are $\mathcal{O}(C)$, with edges from τ_1 to τ_2 of length $d_{\mathcal{I}}(\tau_1, \tau_2)$ where $\mathcal{I}$ is the Q-closure of C. Using such a graph, it is easy to perform $\mathcal{I}$-expansion, choice, and descent in polynomial time, and thus the process of computing the label at ant path π in the solution for a type τ in the constraint set is linear in the length of π, and polynomial in the size of the constraint set. In the case where the diameter of the poset is $\langle 1, 1\rangle$, i.e. we have no subtyping, an obvious optimisation of Helly graph closure is to contract all $\langle 1, 1\rangle$ edges, and we obtain as our solvability criterion that the constraints are unifiable as equalities, and as our closure graph, the usual unification DAG. This observation can also be made for the technique of [33]: our extension of that method allows incorporation of atomic subtyping over Helly posets and of record subtyping without need for $\top$ and $\bot$.

Simplification

It would be interesting to extend the techniques of chapter 3 to deal with constructor types.

In the finite structural case, we can naively compute the shape of each variable α (as in e.g. [18]), and reduce the constraint set to atomic form by substitution. However it is easy to show that in a non-atomic constraint set, substituting an internal variable in this way does not alter the distances between observables which are obtained using the DECOMP rules in addition to DUAL, JOIN and MIN. Since the algorithms for computing tautenings operate only on those distances, in order to test entailment on non-atomic constraint sets it is sufficient to substitute only for the externals. Whilst this can still result in an exponential increase in the size of the constraint set, such would correspond to extremely large observable types, which are rare in practice. This observation is similar to that of [21] where a proof is given that the problem of determining whether a set of constraints entails a single constraint is co-NP complete, even over lattices.

Reduction of a constraint set to atomic form may be relatively expensive, so in practice it is more sensible to use a hybrid approach: to first apply general, easily-validated simplifications (such as moving a negative observable to a least upper bound), and when the constraint set is thus simplified, to reduce to atomic form and H-contract.

However, in order for simplification over Helly posets to be competitive with the approach for lattices of [39], it will be necessary for to extend the techniques to cases where constraints cannot necessarily be reduced to an equivalent atomic form, such as when using regular types, or non-structural subtyping.

Chapter 5

Subtyping and Unification

The polymorphic instance relation can be seen as a kind of subtyping, and one that has received considerable discussion in the literature (recent work includes [51, 31]) since defined by Mitchell in [29]. If we consider the type inference algorithm of ML in a subtyping perspective, substitutions can be viewed as coercions between type scheme, principal types are then minimal types, and principal substitutions are the coercions which must exist to generate a minimal type from the two minimal types combined in an application term.

Suppose we have inferred an argument type scheme σ_1 for a function and we attempt to apply the function to a term with inferred type scheme σ_2. The application will be well-typed exactly if σ_2 is an instance of σ_1. Thus we need to seek a substitution on the free variables of σ_1 and σ_2 and a suitable instantiation of their bound variables, which makes this the case. However, we cannot use just any such substitution: in order to obtain a principal type property, we need to use a *principal* substitution: one which is more general than any other possible substitution we might use.

Although type systems like $F_{\leqslant}$[11] combine subtype and parametric polymorphism, it is hard to see how one might incorporate a decidable instance relation of the sort necessary for ML style type inference, since even in System F the containment relation is undecidable [51]. Instead, we shall start with the predicative fragment of [31] as outlined in chapter 1, which we shall refer to as $ML_{\forall}$, and show how subtyping may be incorporated into that framework.

In this chapter we will define a subtyping system using bounded quantification which combines both kinds of polymorphism, and whose subtyping relation provides a notion of instance suitable for type inference. We will show how the relation can be decided using a method similar to that of the last chapter but with scoping restrictions on possible substitutions involved in removing essential occurrences. The technique bears a considerable similarity to mixed-prefix unification [26] which is used in [31] to generate the principal substitutions.

In chapter 6, we will extend the subtyping system to deal with constrained type schemes, and demonstrate the existence of *factorisations*, analogous to principal substitutions. We shall use these to define a type system with annotations and a type inference algorithm for it which is complete.

In order to simplify the treatment, we will for the moment dispense with the poset of base types Q, so that all our base types, and their subtyping relation, will be introduced in a bounded signature Σ. We will describe in section 5.5 how our results may be generalised to a base poset of atomic types.

5.1 First Class Type Schemes

We will delay the discussion of type inference until chapter 6, and thus for the moment we will not consider constrained quantification. We will deal with the $\rightarrow$ constructor specially, as it is the only constructor which we shall allow to operate on type schemes as well as types. Other constructors, e.g. pairing, can be dealt with using similar techniques, but as we shall see in chapter 6, our type inference system will require us to deal with constructor arguments of polarity `pos` differently to those of polarity `neg`, in that the latter may require annotations on the term constructors which introduce them.

5.1.1 Definition [Types and Type Schemes]:

$$\begin{array}{llcl}
\text{Types} & \tau & ::= & \mathtt{a} \;\mid\; \alpha \;\mid\; \tau_1 \rightarrow \tau_2 \;\mid\; \mathtt{T}\tau_1 \ldots \tau_n \\
\text{Type Schemes} & \sigma & ::= & \tau \;\mid\; \sigma_1 \rightarrow \sigma_2 \;\mid\; \forall \alpha . \sigma \;\mid\; \forall \alpha \leqslant \tau . \sigma
\end{array}$$

Our rules are a combination of two natural notions of subtyping: that given by instantiation, so that $\forall \alpha . \sigma$ is a subtype of $[\tau/\alpha]\sigma$ for any τ, and that given by the consequences of subtype polymorphism. They bear some similarity to the subtyping relation of $\mathrm{F}_{\leqslant}$, although our $\forall$ rules are more in the ML-style, and also to $\mathrm{ML}_{\forall}$, although naturally we have to take the subtyping behaviour of arrow types and constructors into consideration, and we have bounded quantifiers, where it's necessary to check that any instantiation respects the bound. Note that σ is well-formed with respect to Σ if Σ contains all the base types in σ.

Rules for Subtyping

$$\frac{\Sigma^{\uparrow}(\mathtt{a}) = \tau}{\Sigma \vdash \mathtt{a} \leqslant \tau} \qquad (\textsc{Var})$$

$$\frac{\Sigma \vdash \sigma_1 \leqslant \sigma_2 \qquad \Sigma \vdash \sigma_2 \leqslant \sigma_3}{\Sigma \vdash \sigma_1 \leqslant \sigma_3} \tag{\textsc{Trans}}$$

$$\frac{}{\Sigma \vdash \sigma \leqslant \sigma} \tag{\textsc{Taut}}$$

$$\frac{\Sigma \vdash \tau_i \diamond^{\mathtt{T}}{}_i\, \tau_i'}{\Sigma \vdash \mathtt{T}\tau_1 \ldots \tau_n \leqslant \mathtt{T}\tau_1' \ldots \tau_n'} \tag{\textsc{Cons}}$$

$$\frac{\Sigma \vdash \sigma_1' \leqslant \sigma_1 \qquad \Sigma \vdash \sigma_2 \leqslant \sigma_2'}{\Sigma \vdash \sigma_1 \rightarrow \sigma_2 \leqslant \sigma_1' \rightarrow \sigma_2'} \tag{\textsc{Arrow}}$$

$$\frac{}{\Sigma \vdash \forall\alpha.\sigma \leqslant [\tau/\alpha]\sigma} \tag{$\forall$-left}$$

$$\frac{\Sigma, \mathtt{a} \vdash \sigma_1 \leqslant [\mathtt{a}/\alpha]\sigma_2 \qquad \mathtt{a} \text{ does not occur in } \sigma_1 \text{ or } \sigma_2}{\Sigma \vdash \sigma_1 \leqslant \forall\alpha.\sigma_2} \tag{$\forall$-right}$$

$$\frac{\Sigma \vdash \tau_1 \leqslant \tau}{\Sigma \vdash \forall\alpha{\leqslant}\tau.\sigma \leqslant [\tau_1/\alpha]\sigma} \tag{$\forall_{\leqslant}$-left}$$

$$\frac{\Sigma, \mathtt{a}{\leqslant}\tau \vdash \sigma_1 \leqslant [\mathtt{a}/\alpha]\sigma_2 \;\; \mathtt{a} \text{ does not occur in } \sigma_1 \text{ or } \sigma_2}{\Sigma \vdash \sigma_1 \leqslant \forall\alpha{\leqslant}\tau.\sigma_2} \tag{$\forall_{\leqslant}$-right}$$

Reflexivity and Transitivity

Algorithms for testing the subtyping relation (e.g. [11]) typically involve building a derivation of the sequent in question. Usually this involves building an algorithmic set of rules that is equivalent to the original set, and which is syntax directed, so that we always know which was the last algorithmic rule applied to build a sequent. This involves some notion of canonical derivation, in which Trans is redundant, and use of Taut restricted

To develop an algorithm we will construct an equivalent system of rules for which it is the case that all proofs will, if not completely directed, will be directed in the sense that the conclusion of the judgement will determine the last rule applied. Our first step will be to greatly restrict the use of the Taut rule:

5.1.2 Proposition: Suppose we replace the Taut rule in the above proof system with

$$\frac{}{\Sigma \vdash \mathtt{a} \leqslant \mathtt{a}} \tag{\textsc{Refl}}$$

Then for any σ which is well-formed with respect to Σ, $\Sigma \vdash \sigma \leqslant \sigma$

Proof: We will proceed by structural induction on the type σ. If σ is a variable, or the outermost constructor of σ is $\mathtt{T}$ or $\rightarrow$, the result follows from the induction hypothesis. The remaining cases are when the outer constructor is a bounded or unbounded universal quantifier; since the two cases are very similar, we will just do the bounded case. Suppose $\sigma = \forall\alpha\leqslant\tau.\sigma'$, then $\Sigma, \mathtt{a}\leqslant\tau \;\vdash\; [\mathtt{a}/\alpha]\sigma' \leqslant [\mathtt{a}/\alpha]\sigma'$ by the induction hypothesis. So

$$\cfrac{\cfrac{\Sigma, \mathtt{a}\leqslant\tau \;\vdash\; \mathtt{a} \leqslant \tau}{\Sigma \;\vdash\; \mathtt{a}\leqslant\tau \;\vdash\; \forall\alpha\leqslant\tau.\sigma \leqslant [\mathtt{a}/\alpha]\sigma'}}{\Sigma \;\vdash\; \forall\alpha\leqslant\tau.\sigma \leqslant \forall\alpha\leqslant\tau.\sigma}$$

□

In order to generate an algorithmic proof system, it remains to eliminate Trans. We replace the Var rule by

$$\frac{\Sigma \;\vdash\; \Sigma^{\uparrow}(\mathtt{a}) \leqslant \sigma}{\Sigma \;\vdash\; \mathtt{a} \leqslant \sigma} \qquad (\textsc{Prom}^{\uparrow})$$

and $\forall$-left and $\forall_{\leqslant}$-left by

$$\frac{\Sigma \;\vdash\; [\tau/\alpha]\sigma_1 \leqslant \sigma_2}{\Sigma \;\vdash\; \forall\alpha.\sigma_1 \leqslant \sigma_2} \qquad (\forall\text{-LEFT}')$$

$$\frac{\Sigma \;\vdash\; [\tau_1/\alpha]\sigma_1 \leqslant \sigma_2 \qquad \Sigma \;\vdash\; \tau_1 \leqslant \tau}{\Sigma \;\vdash\; \forall\alpha\leqslant\tau.\sigma_1 \leqslant \sigma_2} \qquad (\forall_{\leqslant}\text{-LEFT}')$$

since in the presence of Refl and Trans these rules are equivalent. Further, we shall show that in the presence of these rules, Trans can be completely eliminated.

5.1.3 Definition: We shall call the proof system containing the rules Refl, Prom$^{\uparrow}$, Arrow, Cons, $\forall$-left$'$, $\forall$-right, $\forall_{\leqslant}$-left$'$, $\forall_{\leqslant}$-right, and Trans the *canonical* proof system.

For the rest of this chapter, $\vdash$ will denote provability in the canonical proof system. We will begin with a simple structural lemma, then a limited form of transitivity elimination. We use this to derive a result relating elimination and substitution, and thence full transitivity elimination.

5.1.4 Lemma [Signature permutation]: Suppose that

$$\Sigma = \Sigma_1, \mathtt{a}_1(\leqslant\tau_1), \mathtt{a}_2(\leqslant\tau_2), \Sigma_2$$

and further that

$$\Sigma' = \Sigma_1, \mathtt{a}_2(\leqslant\tau_2), \mathtt{a}_1(\leqslant\tau_1), \Sigma_2$$

is well-formed. If $\Sigma \vdash \sigma_1 \leqslant \sigma_2$, then $\Sigma' \vdash \sigma_1 \leqslant \sigma_2$

Proof: By induction on the structure of the proof of $\Sigma \vdash \sigma_1 \leqslant \sigma_2$. □

5.1.5 Lemma [Renaming]: Suppose $\Sigma \vdash \sigma_1 \leqslant \sigma_2$, $\mathtt{a}$ is a Σ-atom, and $\mathtt{b}$ is an atom that occurs nowhere in the derivation. Then $[\mathtt{b}/\mathtt{a}]\Sigma \vdash [\mathtt{b}/\mathtt{a}]\sigma_1 \leqslant [\mathtt{b}/\mathtt{a}]\sigma_2$.

Proof: By induction on the structure of the proof of $\Sigma \vdash \sigma_1 \leqslant \sigma_2$. □

5.1.6 Lemma [Weakening]: If $\Sigma \vdash \sigma_1 \leqslant \sigma_2$ and $\mathtt{a} \notin \mathrm{dom}(\Sigma)$, then there are derivations with the same structure of $\Sigma, \mathtt{a} \vdash \sigma_1 \leqslant \sigma_2$, and of $\Sigma, \mathtt{a}{\leqslant}\tau \vdash \sigma_1 \leqslant \sigma_2$ for any $\tau \in \tau(\Sigma)$.

Proof: By induction on the structure of the proof of $\Sigma \vdash \sigma_1 \leqslant \sigma_2$, using signature permutation for the case where the last rule in the derivation is a ∀-right rule, and renaming if the situation occurs in a ∀-left rule that the new variable appended to Σ is $\mathtt{a}$. □

5.1.7 Lemma [Partial Transitivity Elimination]: If there are Trans-free derivations $\mathcal{D}_i \equiv \Sigma \vdash \sigma_1 \leqslant \tau$ and $\mathcal{D}_2 \equiv \Sigma \vdash \tau \leqslant \sigma_2$, then there is a Trans-free derivation of $\Sigma \vdash \sigma_1 \leqslant \sigma_2$.

Proof: We proceed by induction on the size (i.e. the number of rule applications) in the derivation. Suppose we have a derivation of size n, and every derivation of size less than n can be rewritten as a Trans-free derivation. If the last rule in the derivation is not Trans, the result is immediate. Otherwise we proceed by case analysis on the rules immediately above Trans. Some combinations are impossible, either because they result in syntactically inconsistent types for τ, or require τ to be a type scheme.

	Refl	Prom$^\uparrow$	Cons	Arrow	∀-l	$\forall_\leqslant$-l	∀-r	$\forall_\leqslant$-r
Refl					×	×		
Prom$^\uparrow$					×	×		
Cons		×		×	×	×		
Arrow		×	×		×	×		
∀-left′					×	×		
$\forall_\leqslant$-left′					×	×		
∀-right	×	×	×	×	×	×	×	×
$\forall_\leqslant$-right	×	×	×	×	×	×	×	×

The cases where one of the derivations is an instance of Refl are trivial, and allowable combinations of Arrow, Cons, and Prom$^{\uparrow}$ are straightforward using the induction hypothesis. The remaining cases are when $\forall$-left$'$ ends $\mathcal{D}_1$, or $\forall$-right ends $\mathcal{D}_2$. With $\forall$-left on the left we have

$$\dfrac{\dfrac{\Sigma \vdash [\tau_2/\alpha]\sigma \leqslant \tau \qquad \Sigma \vdash \tau_2 \leqslant \tau_1}{\Sigma \vdash \forall\alpha{\leqslant}\tau_1.\sigma \leqslant \tau} \qquad \Sigma \vdash \tau \leqslant \sigma_1}{\Sigma \vdash \forall\alpha{\leqslant}\tau_1.\sigma \leqslant \sigma_1}$$

becomes

$$\dfrac{\dfrac{\Sigma \vdash [\tau_2/\alpha]\sigma \leqslant \tau \qquad \Sigma \vdash \tau \leqslant \sigma_1}{\Sigma \vdash [\tau_2/\alpha]\sigma \leqslant \sigma_1} \qquad \Sigma \vdash \tau_2 \leqslant \tau_1}{\Sigma \vdash \forall\alpha{\leqslant}\tau_1.\sigma \leqslant \sigma_1}$$

and with of $\forall_{\leqslant}$-right on the right

$$\dfrac{\Sigma \vdash \sigma \leqslant \tau \qquad \dfrac{\Sigma, \mathsf{a}{\leqslant}\tau_1 \vdash \tau \leqslant [\mathsf{a}/\alpha]\sigma_1 \quad (\mathsf{a} \text{ does not occur in } \tau \text{ or } \sigma_1)}{\Sigma \vdash \tau \leqslant \forall\alpha{\leqslant}\tau_1.\sigma_1}}{\Sigma \vdash \sigma \leqslant \forall\alpha{\leqslant}\tau_1.\sigma_1}$$

which becomes

$$\dfrac{\dfrac{\Sigma, \mathsf{a}{\leqslant}\tau_1 \vdash \sigma \leqslant \tau \qquad \Sigma, \mathsf{a}{\leqslant}\tau_1 \vdash \tau \leqslant [\mathsf{a}/\alpha]\sigma_1}{\Sigma, \mathsf{a} \leqslant \tau_1 \vdash \sigma \leqslant [\mathsf{a}/\alpha]\sigma_1} \qquad \mathsf{a} \text{ does not occur in } \sigma \text{ or } \sigma_1}{\Sigma \vdash \sigma \leqslant \forall\alpha{\leqslant}\tau_1.\sigma_1}$$

the use of weakening and the alteration of the side-condition from a not occurring in τ to a not occurring in σ being justified by the fact that since a does not occur in Σ, nor can it in σ.

In either case, we have a use of transitivity at the base of a derivation of smaller size, so we can use the induction hypothesis. (The cases for $\forall$-left on the left and $\forall$-right on the right are similar.) □

5.1.8 Definition: Suppose $\Sigma = \Sigma_1, \mathsf{a}({\leqslant}\tau_1), \Sigma_2$, and we have $\tau \in \tau(\Sigma_1)$ such that if $\Sigma^{\uparrow}(\mathsf{a}){\downarrow}$ then $\Sigma_1 \vdash \tau \leqslant \tau_1$ by a Trans-free derivation. We define $\langle\tau/\mathsf{a}\rangle\Sigma$ to be $\Sigma_1, [\tau/\alpha]\Sigma_2$.

5.1.9 Lemma: Suppose there exists a TRANS-free derivation of $\Sigma \;\vdash\; \sigma_1 \leqslant \sigma_2$. Then for any τ such that $\langle\tau/\mathtt{a}\rangle\Sigma$ is defined, there is a TRANS-free derivation of $\langle\tau/\mathtt{a}\rangle\Sigma \;\vdash\; [\tau/\mathtt{a}]\sigma_1 \leqslant [\tau/\mathtt{a}]\sigma_2$.

Proof: Consider a derivation for some $\Sigma \;\vdash\; \sigma_1 \leqslant \sigma_2$, such that the result holds for all derivations of lesser height. We proceed by cases on the last rule in the derivation.

REFL Straightforward

ARROW by the induction hypothesis.

CONS by the induction hypothesis.

PROM$^{\uparrow}$ If the base type σ_1 is other than $\mathtt{a}$, then it follows from the induction hypothesis that there is a TRANS-free proof of $\langle\tau/\mathtt{a}\rangle\Sigma \;\vdash\; [\tau/\mathtt{a}](\Sigma^{\uparrow}(\mathtt{b})) \leqslant [\tau/\mathtt{a}]\sigma_2$, and thus since $(\langle\tau/\mathtt{a}\rangle\Sigma)(\mathtt{b}) = [\tau/\mathtt{a}](\Sigma^{\uparrow}(\mathtt{b})$, there is one of $\langle\tau/\mathtt{a}\rangle\Sigma \;\vdash\; \mathtt{b} \leqslant [\tau/\mathtt{a}]\sigma_2$.

So suppose $\sigma_1 = \mathtt{a}$, and that $\Sigma^{\uparrow}(\mathtt{a}) = \tau'$. Then the rule contains the premise $\Sigma \;\vdash\; \tau' \leqslant \sigma_2$, and so by the induction hypothesis (and since $\mathtt{a}$ does not occur in τ') we have a proof of $\langle\tau/\mathtt{a}\rangle\Sigma \;\vdash\; \tau' \leqslant [\tau/\mathtt{a}]\sigma_2$. We can apply lemma(5.1.7) to this together with the proof of $\Sigma_1 \;\vdash\; \tau \leqslant \tau'$ to obtain a TRANS-free proof of the required judgement.

$\forall_{\leqslant}$-LEFT Suppose the last rule is

$$\frac{\Sigma \;\vdash\; [\tau_2/\beta]\sigma_1 \leqslant \sigma_2 \qquad \Sigma \;\vdash\; \tau_2 \leqslant \tau_1}{\Sigma \;\vdash\; \forall\beta{\leqslant}\tau_1.\sigma_1 \leqslant \sigma_2}$$

By the induction hypothesis, we have proofs of $\langle\tau/\mathtt{a}\rangle\Sigma \;\vdash\; [\tau/\mathtt{a}][\tau_2/\beta]\sigma_1 \leqslant [\tau/\mathtt{a}]\sigma_2$ and $\langle\tau/\mathtt{a}\rangle\Sigma \;\vdash\; [\tau/\mathtt{a}]\tau_1 \leqslant [\tau/\mathtt{a}]\tau_1$. But since $[\tau/\alpha][\tau_2/\beta]\sigma_1 = [([\tau/\alpha]\tau_2)/\beta][\tau/\alpha]\sigma_1$, we may apply $\forall_{\leqslant}$-LEFT to obtain the required proof.

$\forall$-LEFT Similarly.

$\forall_{\leqslant}$-RIGHT Suppose the last rule is

$$\frac{\Sigma, \mathtt{b}{\leqslant}\tau_2 \;\vdash\; \sigma_1 \leqslant [\mathtt{b}/\beta]\sigma_2 \qquad \mathtt{b} \text{ does not occur in } \sigma_1}{\Sigma \;\vdash\; \sigma_1 \leqslant \forall\beta{\leqslant}\tau_2.\sigma_2}$$

since $\mathtt{a}$ occurs in Σ, $\mathtt{b} \neq \mathtt{a}$. Then by the induction hypothesis we have a proof of $\langle\tau/\mathtt{a}\rangle\Sigma, \mathtt{b}{\leqslant}[\tau/\mathtt{a}]\tau_2 \;\vdash\; [\tau/\mathtt{a}]\sigma_1 \leqslant [\tau/\mathtt{a}][\mathtt{b}/\beta]\sigma_2$. But since β does not occur in τ, $[\tau/\mathtt{a}][\mathtt{b}/\beta]\sigma_2 = [\mathtt{b}/\beta][\tau/\mathtt{a}]\sigma_2$, so an application of $\forall_{\leqslant}$-RIGHT suffices.

$\forall$-RIGHT Similarly. □

5.1.10 Lemma [Full transitivity]: if there is a derivation of $\Sigma \vdash \sigma_1 \leqslant \sigma_2$, then there is a TRANS-free derivation.

Proof: Suppose we have a derivation of $\Sigma \vdash \sigma_1 \leqslant \sigma_2$ where the final rule is TRANS with mediating type σ_3, and the subderivations are TRANS-free. We define an ordering on such derivations to be (q, s) ordered lexicographically, where q is the depth of quantifier nesting of σ_3 and s is the size of the derivation. Suppose all such derivations of measure less than (q, s) can be rewritten to be transitivity-free.

Consider the possibilities for the last rule in the subderivations: almost all cases are straightforward using the induction hypothesis in the manner of lemma(5.1.7). The interesting combinations are when we have $\forall_{\leqslant}$-RIGHT on the left and $\forall_{\leqslant}$-LEFT on the right, or $\forall$-RIGHT on the left and $\forall$-LEFT on the right, the two cases being essentially similar. Suppose then the rule is

$$\dfrac{\dfrac{\Sigma, \mathsf{a}\leqslant\tau_3 \vdash \sigma_1 \leqslant [\mathsf{a}/\alpha]\sigma_3}{\Sigma \vdash \sigma_1 \leqslant \forall\alpha\leqslant\tau_3.\sigma_3} \quad \dfrac{\Sigma \vdash [\tau/\alpha]\sigma_3 \leqslant \sigma_2 \quad \Sigma \vdash \tau \leqslant \tau_3}{\Sigma \vdash \forall\alpha\leqslant\tau_3.\sigma_3 \leqslant \sigma_2}}{\Sigma \vdash \sigma_1 \leqslant \sigma_2}$$

where a does not occur in σ_1 or σ_3

Now $\langle\tau/\mathsf{a}\rangle\Sigma = \Sigma$, so we may apply lemma(5.1.9) to obtain a TRANS-free derivation of $\Sigma \vdash [\tau/\mathsf{a}]\sigma_1 \leqslant [\tau/\mathsf{a}][\mathsf{a}/\alpha]\sigma_3$. But since a does not occur in σ_1, this is just $\Sigma \vdash \sigma_1 \leqslant [\tau/\alpha]\sigma_3$. Now we may rewrite the derivation above as

$$\dfrac{\Sigma \vdash \sigma_1 \leqslant [\tau/\alpha_3]\sigma_3 \quad \Sigma \vdash [\tau/\alpha]\sigma_3 \leqslant \sigma_2}{\Sigma \vdash \sigma_1 \leqslant \sigma_2}$$

where the subderivations to the TRANS are TRANS-free, and the size of the rule is $(q-1, n')$ since the depth of quantifier nesting in $[\tau/\alpha]\sigma_3$ is less than that of $\forall\alpha\leqslant\tau_3.\sigma_3$. The result follows from the induction hypothesis. □

5.1.11 Definition [Canonical Derivations]: A derivation in the canonical proof system is *canonical* if it contains no instance of TRANS, and no instance of $\forall$-RIGHT or $\forall_{\leqslant}$-RIGHT directly above an instance of PROM$^{\uparrow}$, $\forall$-LEFT$'$, or $\forall_{\leqslant}$-LEFT$'$.

5.1.12 Theorem: Suppose there is a derivation of $\Sigma \vdash \sigma_1 \leqslant \sigma_2$. Then there is a canonical derivation.

Proof: By transitivity elimination, then simple permutation of the rules. □

The notable property of a canonical derivation is that the choice of the last rule in a canonical derivation of $\Sigma \vdash \sigma_1 \leqslant \sigma_2$ is exactly determined by σ_1 and σ_2. However, this does not enable us to derive the premises of the rule in all cases; in particular, in the algorithmic equivalent of the $\forall$-LEFT rules, we are unable to derive the substitution term. A similar phenomenon is exhibited, for example, in the typed λ-calculus, where the type of the bound variable in an application term is lost.

Algorithmic Subtyping

In order to perform type inference in the ML style, we need to be able to address the question of how we might decide the subtype relation between two terms containing free variables. Suppose we have $\Sigma = \mathtt{Nat}, \mathtt{Int} \leqslant \mathtt{Nat}$, the assertion

$$\Sigma \vdash \forall\alpha.\gamma{\to}\alpha \leqslant \forall\beta{\leqslant}\mathtt{Nat}.\beta{\to}\mathtt{Int}{\to}\beta$$

with γ a free variable which can be instantiated with any type in $\tau(\Sigma)$. Now if there is a substitution τ_γ for γ in $\tau(\Sigma)$ such that the above holds, using our knowledge about canonical derivations, the last rule in such a derivation must be $\forall_\leqslant$-RIGHT, whose premise must be

$$\Sigma, \mathtt{b}{\leqslant}\mathtt{Nat} \vdash \forall\alpha.\tau_\gamma{\to}\alpha \leqslant \mathtt{b}{\to}\mathtt{Int}{\to}\mathtt{b}$$

and the one preceding that must be $\forall$-LEFT$'$. Suppose the substitution used in that rule for α is $\tau_\alpha \in \tau(\Sigma, \mathtt{b}{\leqslant}\mathtt{Nat})$, then the premise is

$$\Sigma, \mathtt{b}{\leqslant}\mathtt{Nat} \vdash \tau_\gamma{\to}\tau_\alpha \leqslant \mathtt{b}{\to}\mathtt{Int}{\to}\mathtt{b}$$

Now the preceding rule must be ARROW, so we have two sequents.

$$\Sigma, \mathtt{b}{\leqslant}\mathtt{Nat} \vdash \mathtt{b} \leqslant \tau_\gamma \quad \text{and} \quad \Sigma, \mathtt{b}{\leqslant}\mathtt{Nat} \vdash \tau_\alpha \leqslant \mathtt{Int}{\to}\mathtt{b}$$

Let us consider the first of these. Since τ_γ cannot be b, (by scoping restrictions) the last rule in the derivation must be a PROM$^\uparrow$ rule. So we must have $\Sigma, \mathtt{b}{\leqslant}\mathtt{Nat} \vdash \mathtt{Nat} \leqslant \tau_\gamma$. Such a derivation cannot involve b, so it must be the case that $\Sigma \vdash \mathtt{Nat} \leqslant \tau_\gamma$.

Now we consider the second. By syntactic examination of the rules, the last one can only be Prom$^{\uparrow}$ (if τ_α is a Σ-atom) or Arrow (if τ_α is an arrow type). Now suppose τ_α is a Σ-atom, then after a sequence of promotions, we must reach an arrow type, say $\tau_1{\rightarrow}\tau_2$, with the property that $\Sigma, \mathtt{b}{\leqslant}\mathtt{Nat} \;\vdash\; \tau_2 \leqslant \mathtt{b}$, So τ_2 must be $\mathtt{b}$. Therefore, we must have in Σ an atom whose bound is of the form $\tau_1{\rightarrow}\mathtt{b}$, which is impossible by well-formedness of $\Sigma, \mathtt{b}{\leqslant}\mathtt{Nat}$, so τ_α must be an arrow type. Substituting $\tau_1{\rightarrow}\tau_2$ for α, and decomposing using an arrow rule, we have

$$\Sigma, \mathtt{b}{\leqslant}\mathtt{Nat} \;\vdash\; \mathtt{Int} \leqslant \tau_1 \quad \text{and} \quad \Sigma, \mathtt{b}{\leqslant}\mathtt{Nat} \;\vdash\; \tau_2 \leqslant \mathtt{b}$$

Evidently the only possibility for τ_2 is $\mathtt{b}$. And for any value of τ_1 such that $\Sigma, \mathtt{b}{\leqslant}\mathtt{Nat} \;\vdash\; \mathtt{Int} \leqslant \tau_1$, we have $\Sigma \;\vdash\; \mathtt{Int} \leqslant \tau_1$. So the inequality holds iff we can find τ_γ, τ_1 with the property that $\Sigma \;\vdash\; \mathtt{Nat} \leqslant \tau_\gamma$ and $\Sigma \;\vdash\; \mathtt{Int} \leqslant \tau_1$.

In order to regularise this kind of reasoning, we need a way of dealing with the variables which appear in several sequents, and thus a way of reasoning with many sequents at once. To proceed more formally, we introduce *quantified subtype problems*, similar to unification problems. The above inequality is represented by the QSP

$$\forall\mathtt{Int}\forall\mathtt{Nat}{\leqslant}\mathtt{Int}\exists\gamma\forall\mathtt{b}{\leqslant}\mathtt{Nat}\exists\alpha.\, \gamma{\rightarrow}\alpha{\leqslant}\mathtt{b}{\rightarrow}\mathtt{Int}{\rightarrow}\mathtt{b}$$

In order to deal with sequents that contain variables, we must modify slightly our notion of what constitutes a well-formed sequent: we will now allow types to contain free variables (i.e. variables not occurring in the domain of the signature), but in order to ensure that substitution instances of such sequents are well-formed, we annotate free variables with an initial (non-empty) segment of the signature. If such a variable occurs in Σ, it may only do so after the segment with which it is annotated.

Given a sequent $\Sigma \;\vdash\; \tau_1 \leqslant \tau_2$, a well-scoped atomic substitution $[\tau/\alpha_{\Sigma'}]$ will be such that any variable in τ is either in $\mathrm{dom}(\Sigma')$, or $\beta_{\Sigma''}$ for some Σ'' a non-empty initial segment of Σ'. A well-scoped substitution will be one composed of well-scoped atomic substitutions.

We will also define substitutions on base types: for a sequent $\mathcal{D} \equiv \Sigma \;\vdash\; \tau_1 \leqslant \tau_2$, if $\langle\tau/\mathtt{a}\rangle$ is defined, we define $[\tau/\mathtt{a}]\mathcal{D} = \langle\tau/\mathtt{a}\rangle\Sigma \;\vdash\; [\tau/\mathtt{a}]\tau_1 \leqslant [\tau/\mathtt{a}]\tau_2$.

The scoping issues here are more complicated than mixed-prefix unification. In that case, a scope check occurs exactly if some existentially bound variable is bound to a term containing a universally quantified variable such that the existential quantifier is not within the scope of the universal. In the current situation, it may be possible to resolve such an occurrence by promotion (such as in the $\mathtt{b} \leqslant \tau_\gamma$ case) and thus a scope check will not occur.

5.2 Leaves and Essential Occurrences

Before proceeding, we shall investigate further properties of the subtyping system which generalise and strengthen the ad-hoc deductions used above. Our first action will be to decompose an inequality between type schemes into inequalities between types, creating new variables to correspond to substituted values in $\forall$-LEFT$'$ rules. Once we have done this, our attention naturally focuses on the quantification-free fragment of the proof system.

5.2.1 Remark: Suppose we have a derivation of $\Sigma \vdash \tau_1 \leqslant \tau_2$, then we have a TRANS-free derivation. A simple induction suffices to show that this derivation uses only the rules REFL, CONS, PROM$^{\uparrow}$, TRANS, and the ARROW

Our strategy proceeded by removing the binding $\mathtt{b}{\leqslant}\mathtt{Nat}$ from the signature, by solving for those variables which had to contain $\mathtt{b}$, and promoting $\mathtt{b}$ to $\mathtt{Nat}$ in other constraints. We were able to deduce from $\Sigma, \mathtt{b}{\leqslant}\mathtt{Nat} \vdash \tau_2 \leqslant \mathtt{b}$ that τ_2 could only take the value $\mathtt{b}$. This was possible because $\mathtt{b}$ was guaranteed not to occur as the bound of another Σ-atom. We will call a Σ-atom with this property a *leaf.* We also used the result that if $\tau \leqslant \tau'$, and τ' is a constructor type in which $\mathtt{b}$ occurs positively, then τ must also be a constructor type in which $\mathtt{b}$ occurs positively. We will generalise this latter result to occurrences of $\mathtt{b}$ which we shall call *essential.*

5.2.2 Definition: $\mathtt{a} \in \text{atom}(\Sigma)$ is a *leaf* if it does not occur in the range of Σ. We write $\Sigma\backslash_{\mathtt{a}}$ for Σ with the binding for $\mathtt{a}$ removed; it is easy to see that if Σ is a valid signature, so is $\Sigma\backslash_{\mathtt{a}}$.

The following are straightforward:

5.2.3 Proposition: Suppose $\mathtt{a}$ is a leaf in Σ

Weakening If $\Sigma\backslash_{\mathtt{a}} \vdash \tau_1 \leqslant \tau_2$, then $\Sigma \vdash \tau_1 \leqslant \tau_2$.

Strengthening If $\Sigma \vdash \tau_1 \leqslant \tau_2$, and $\mathtt{a}$ does not occur in τ_1 or τ_2, then $\Sigma\backslash_{\mathtt{a}} \vdash \tau_1 \leqslant \tau_2$.

5.2.4 Definition [Type Shape and Size]: We need to extend the notion of type shape to deal with non-atomic base types.

$$\begin{array}{lcll}
\mathcal{S}(\mathtt{a}) & = & \mathtt{a} & \text{if } \mathtt{a} \in \text{atom}(\Sigma) \text{ and } \Sigma^{\uparrow}(\mathtt{a})\uparrow \\
\mathcal{S}(\mathtt{a}) & = & \mathcal{S}(\Sigma^{\uparrow}(\mathtt{a})) & \text{if } \Sigma^{\uparrow}(\mathtt{a}) = \tau \\
\mathcal{S}(\mathtt{T}\tau_1 \ldots \tau_n) & = & \mathtt{T}\,\mathcal{S}(\tau_1) \ldots \mathcal{S}(\tau_n) &
\end{array}$$

In order to show this definition terminates, we will use a size measure on types, an argument similar to that of [8]. Since we will use this definition later, it is given in a slightly more general form than is required here. Let Σ be a bounded signature.

$$\begin{array}{lcll}
\text{size}_\Sigma(\alpha) & = & 1 & \\
\text{size}_\Sigma(\mathtt{a}) & = & 1 & \text{if } \Sigma^{\uparrow}(\mathtt{a})\uparrow \text{ or } \mathtt{a} \in Q \\
\text{size}_\Sigma(\mathtt{a}) & = & \text{size}_\Sigma(\tau) & \text{if } \Sigma^{\uparrow}(\mathtt{a}) = \tau \\
\text{size}_\Sigma(\mathtt{T}\tau_1 \ldots \tau_n) & = & 1 + \text{size}_\Sigma(\tau_1) + \ldots + \text{size}_\Sigma(\tau_n) &
\end{array}$$

5.2.5 Proposition: the computation of $\mathcal{S}_C(\tau)$ terminates.

Proof: We define the *index* of $i(\mathtt{a})$ in Σ to be the number of bindings prior to $\mathtt{a}$ in Σ. We define $m(\tau)$ to be $(i(\tau), \text{size}_\Sigma(\tau))$, with the usual lexicographic ordering on $\mathbb{N} \times \mathbb{N}$. Each step in the computation of $\mathcal{S}(\tau)$ decreases $m(\tau)$. □

5.2.6 Remark: If $\Sigma \vdash \tau_1 \leqslant \tau_2$, then $\mathcal{S}(\tau_1) = \mathcal{S}(\tau_2)$.

We will assume the constraint set is shape consistent – that is, that there is a substitution $\theta_S : \text{fv}(c) \rightarrow \tau(\text{atom}(\Sigma))$ such that $\mathcal{S}(\theta_S \tau_1) = \mathcal{S}(\theta_S \tau_2)$ for any $\tau_1 \leqslant \tau_2 \in C$. The existence of such a substitution is clearly a necessary condition for solvability.

5.2.7 Definition [Essential Occurrence]: Suppose $\mathtt{a}$ is a leaf in Σ. We define $\mathtt{a}$ to *occur essentially* in $\tau_1 \leqslant \tau_2$ if

- $\Sigma^{\uparrow}(\mathtt{a})\downarrow$ and $\mathtt{a}$ occurs in τ_1 other than positively (i.e. with positive or mixed polarity), or $\mathtt{a}$ occurs in τ_2 other than negatively.
- $\Sigma^{\uparrow}(\mathtt{a})\uparrow$ and $\mathtt{a}$ occurs in τ_1 or τ_2.

5.2.8 Lemma:

1. If $\Sigma \vdash \mathtt{a} \leqslant \tau$ and $\mathtt{a}$ occurs in τ, then the proof is an instance of Refl.
2. If $\Sigma \vdash \tau_1 \leqslant \tau_2$, and $\mathtt{a}$ occurs essentially in $\tau_1 \leqslant \tau_2$, Then either the last rule in the derivation (and thus the whole derivation) is an instance of Refl, or $\mathtt{a}$ occurs essentially in one of the rule's premises.
3. Suppose $\Sigma \vdash \tau_1 \leqslant \tau_2$, and $\mathtt{a}$ occurs essentially in $\tau_1 \leqslant \tau_2$. Then $\mathtt{a}$ occurs in each of τ_1 and τ_2.

Proof:

1. τ cannot be of the form $\mathtt{T}\overline{\tau}$, since it contains $\mathtt{a}$, and we must have $\mathcal{S}(\mathtt{a}) = \mathcal{S}(\tau)$. So τ must be a base type, and since $\mathtt{a}$ occurs in τ, $\tau = \mathtt{a}$. Then obviously the last rule in the derivation cannot be Cons. Now suppose the last rule in the proof is Prom$^{\uparrow}$, then the premise of this rule cannot be of the form $\Sigma \ \vdash \ \mathtt{T}\overline{\tau} \leqslant \mathtt{a}$, since no rule has a consequence of this form, so it must be of the form $\Sigma \ \vdash \ \mathtt{b} \leqslant \mathtt{a}$. And similarly, if the previous rule is an instance of Prom$^{\uparrow}$, then *its* premise must be of the form $\Sigma \ \vdash \ \mathtt{c} \leqslant \mathtt{a}$.

   So by induction the proof must consist of a sequence of zero or more instances of Prom$^{\uparrow}$ below an instance of Refl, and for each Prom$^{\uparrow}$ rule, the variable on the LHS of its premise must occur earlier in Σ than the variable on the LHS of its consequent. But the variable immediately below the instance of Refl must have $\mathtt{a}$ as its bound, so must occur later in Σ than $\mathtt{a}$. So if there are any instances of Prom$^{\uparrow}$ we have an immediate contradiction.

2. If the last rule is Cons, and $\tau_1 = \mathtt{T}\overline{\tau}$, the result is straightforward by case analysis on the polarity of $\mathtt{T}$ in each of its arguments.

   So suppose the last rule is Prom$^{\uparrow}$. If $\tau_1 = \mathtt{a}$ then we must have $\Sigma^{\uparrow}(\mathtt{a})\downarrow$, so an essential occurrence is exactly an occurrence in τ_2 with positive or mixed polarity, but by part 1, $\mathtt{a}$ cannot occur in τ_2 unless the derivation is an instance of Refl. So τ_1 must be some other variable $\mathtt{b}$, in which case, $\mathtt{a}$ must occur positively or with mixed polarity in τ_2, and thus it occurs essentially in $\Sigma^{\uparrow}(\mathtt{b}) \leqslant \tau_2$.

3. We proceed by induction on the proof of $\Sigma \ \vdash \ \tau_1 \leqslant \tau_2$. Suppose the proposition is true for each of the premises in the last rule. We proceed by cases

   Refl Trivially true.

   Prom$^{\uparrow}$ We will show that the last rule cannot be an instance of Prom$^{\uparrow}$: suppose it were, then $\tau_1 = \mathtt{b}$, and $\mathtt{a} \neq \mathtt{b}$ by part 1. By the induction hypothesis, if $\mathtt{a}$ occurs essentially in $\Sigma^{\uparrow}(\mathtt{b}) \leqslant \tau_2$ it must occur in $\Sigma^{\uparrow}(\mathtt{b})$. But this cannot be the case since $\mathtt{a}$ is a leaf.

   Cons If $\mathtt{a}$ occurs essentially in $\mathtt{T}\overline{\tau}_1 \leqslant \mathtt{T}\overline{\tau}_2$, then by part 2, there must be some premise $\tau_1 \leqslant \tau_2$ in which it occurs essentially. Thus by the induction hypothesis it occurs in each of τ_1 and τ_2, and thus in each of $\mathtt{T}\overline{\tau}_1$ and $\mathtt{T}\overline{\tau}_2$ □

5.2.9 Corollary: Suppose $\Sigma \ \vdash \ \tau_1 \leqslant \tau_2$, and $\mathtt{a}$ occurs essentially in $\tau_1 \leqslant \tau_2$. Then

1. if one of τ_1 and τ_2 is $\mathtt{a}$, so is the other.

2. if one of τ_1 and τ_2 has the form $\mathtt{T}\tau'_1 \ldots \tau'_n$, then the other has the form $\mathtt{T}\tau''_1 \ldots \tau''_n$.

Proof: Suppose $\mathtt{a}$ occurs essentially in $\tau_1 \leqslant \tau_2$. Then from lemma(5.2.8(3)) we know that $\mathtt{a}$ must occur in each of τ_1 and τ_2. Now

1. is immediate since the derivation must be an instance of Refl.

2. Note that the only other syntactically allowable possibility from the form of the rules is that τ_1 is a base type and τ_2 not, with the last rule in the derivation being Prom$^{\uparrow}$. But since $\mathtt{a}$ occurs in τ_1, we must have $\tau_1 = \mathtt{a}$, with $\Sigma^{\uparrow}(\mathtt{a})\downarrow$, and so by lemma(5.2.8(1)) $\mathtt{a}$ does not occur in τ_2. So, since $\Sigma^{\uparrow}(\mathtt{a})\downarrow$ and $\mathtt{a}$ occurs only positively in τ_1 and not at all in τ_2, it does not occur essentially in $\tau_1 \leqslant \tau_2$, contradicting the hypothesis. □

5.3 Quantified Subtype Problems

Quantified subtype problems (QSPs) bear a strong similarity to first order mixed prefix unification problems. The essential differences are that we have inequality instead of equality, that we incorporate a bounded universal quantifier, and that a quantifier prefix is replaced by a more general form of quantifier nesting. The last difference is not strictly necessary, however: we simply adopt the more flexible format in order to avoid generating additional solutions to inequalities such as e.g. $\beta \leqslant \beta$ which might be created by moving quantifiers outwards.

We shall assume all the problems we work with are closed, i.e. every variable is in the scope of either an existential or universal quantifier which binds it, and that they begin with an unbounded universal quantifier. The latter restriction can easily be relaxed, but we shall ignore here the question of an underlying order on atomic types, and consider only the order defined by Σ.

QSPs have the following syntax:

$$\Pi \;::=\; \top \;\mid\; \tau_1 \leqslant \tau_2 \;\mid\; \Pi \wedge \Pi \ldots \wedge \Pi \;\mid\; \forall \mathtt{a}.\Pi \;\mid\; \forall \mathtt{a}{\leqslant}\tau.\Pi \;\mid\; \exists \alpha.\Pi$$

where $\wedge$ represents conjunction, and $\top$ the solved problem.

It will be appropriate to view a QSP as a notation for a set of sequents for inequalities between types. For example

$$\forall \mathtt{Int}\, \forall \mathtt{Nat}{\leqslant}\mathtt{Int}\, \exists \alpha\, \forall \mathtt{a}{\leqslant}\mathtt{Nat}\, \exists \beta\, .\beta{\leqslant}\alpha \;\wedge\; \mathtt{Int}{\leqslant}\beta \;\wedge\; \beta{\leqslant}\mathtt{a}$$

can be viewed (writing $\Sigma_1 = \mathtt{Int}, \mathtt{Nat}{\leqslant}\mathtt{Int}$ and $\Sigma_2 = \Sigma_1, \mathtt{a}{\leqslant}\mathtt{Nat}$) as representing the sequents

$$\begin{array}{l}
\Sigma_2 \ \vdash \ \beta_{\Sigma_2} \leqslant \alpha_{\Sigma_1} \\
\Sigma_2 \ \vdash \ \mathtt{Int} \leqslant \beta_{\Sigma_2} \\
\Sigma_2 \ \vdash \ \beta_{\Sigma_2} \leqslant \mathtt{a}
\end{array}$$

We define the set of sequents $S(\Pi)$ corresponding to a subtyping problem to be $S(\Pi)_\bullet$, where $S(\Pi)_\Sigma$ is as follows:

$$\begin{array}{lcl}
S(\top)_\Sigma & = & \varnothing \\
S(\tau_1{\leqslant}\tau_2)_\Sigma & = & \Sigma \ \vdash \ \tau_1 \leqslant \tau_2 \\
S(\Pi_1 \wedge \ldots \wedge \Pi_n)_\Sigma & = & S(\Pi_1)_\Sigma \cup \ldots \cup S(\Pi_n)_\Sigma \\
S(\forall \mathtt{a}.\Pi)_\Sigma & = & S(\Pi)_{\Sigma,\mathtt{a}} \\
S(\forall \mathtt{a}{\leqslant}\tau.\Pi)_\Sigma & = & S(\Pi)_{\Sigma,\mathtt{a}\leqslant\tau} \\
S(\exists\alpha.\Pi)_\Sigma & = & S([\alpha_\Sigma/\alpha]\Pi)_\Sigma
\end{array}$$

When we wish to refer to a subproblem Π' occurring in a context P within Π, we will write $\Pi = P\{\Pi'\}$. We will write Σ_P for the environment built up from the universal quantifiers in P in whose scope is the "hole" in which Π occurs, and Σ_Π to denote the environment corresponding to a the context in which Π occurs – (the two notions differ in the case of e.g. $P\{\forall\alpha.\Pi\}$)

5.3.1 Definition [Substitution]: For α existentially quantified in Π, we define the atomic substitution $[\tau/\alpha]\Pi$ as follows: let $\overline{\beta}$ be the set of variables in τ not bound in Π. Then $[\tau/\alpha]\Pi$ is the problem obtained by substituting τ for α in Π and replacing the existential quantifier for α by existential quantifiers for $\overline{\beta}$. Such an operations is *well-scoped* if every occurrence of β is within the scope of every variable in τ which occurs in Π.

For example, $[\gamma{\rightarrow}\mathtt{a}/\beta]$ is well-scoped for the problem

$$\forall \mathtt{a} \exists \beta . \beta \leqslant \mathtt{a}{\rightarrow}\mathtt{a}$$

and substitution yields the problem

$$\forall \mathtt{a}.\exists\gamma.\gamma{\rightarrow}\mathtt{a} \leqslant \mathtt{a}{\rightarrow}\mathtt{a}$$

For $\mathtt{a}$ universally quantified, if $\Pi = P\{\forall \mathtt{a}(\leqslant \tau).\Pi_1\}$, we define $[\tau'/\mathtt{a}]\Pi$ to be $P\{\Pi_1'\}$, where Π_1' is obtained by replacing all occurrences of α by τ. We will only allow substitutions such that $[\tau/\mathtt{a}]\Pi$ is well-formed.
A substitution θ on Π is a composition of well-scoped atomic substitutions.

5.3.2 Proposition: Suppose θ is a substitution on Π. Then θ is well-formed on the sequents in $S(\Pi)$, and $(S(\theta\Pi)) = \theta(S(\Pi))$.

5.3.3 Definition [Solution of a QSP]: A *solution* θ of a QSP Π is a ground substitution such that all the sequents in $S(\theta\Pi)$ are provable.

For Example, the QSP

$$\forall\mathtt{Int}\forall\mathtt{Nat}\leqslant\mathtt{Int}\exists\gamma\forall\mathtt{a}\leqslant\mathtt{Nat}\exists\alpha.\gamma\to\alpha\leqslant\mathtt{a}\to\mathtt{Int}\to\mathtt{a}$$

has solutions $[\mathtt{Int}\to\mathtt{a}/\alpha]\circ[\mathtt{Int}/\gamma]$ and $[\mathtt{Int}\to\mathtt{a}/\alpha]\circ[\mathtt{Nat}/\gamma]$.

5.4 Transitions

We will solve QSPs by using transitions: each transition will preserve well-formedness of the QSP. There are three kinds of transitions: *flattening*, *binding*, and *projection*. Flattening is the process of applying all the possible decompositions between types whose applicability is immediately obvious from the form of the types in question. Binding uses corollary(5.2.9) to justify variable substitutions which simplify the problem, and binding removes redundant quantifiers from the problem.

Flattening We define $\Pi^\flat = \Pi^\flat_\bullet$.

$$\begin{array}{lcll}
(\forall\mathtt{a}.\Pi)^\flat_\Sigma & = & \forall\mathtt{a}.(\Pi)^\flat_{\Sigma,\mathtt{a}} & \\
(\forall\mathtt{a}\leqslant\tau.\Pi)^\flat_\Sigma & = & \forall\mathtt{a}\leqslant\tau.(\Pi)^\flat_{\Sigma,\mathtt{a}\leqslant\tau} & \\
(\exists\alpha.\Pi)^\flat_\Sigma & = & \exists\alpha.(\Pi)^\flat_\Sigma & \\
(\Pi_1\wedge\ldots\wedge\Pi_n)^\flat_\Sigma & = & (\Pi_1)^\flat_\Sigma\wedge\ldots\wedge(\Pi_n)^\flat_\Sigma & \\
 & & & \\
(\tau\leqslant\tau)^\flat_\Sigma & = & \top & \\
 & & & \\
(\mathtt{T}\tau_1\ldots\tau_n\leqslant\mathtt{T}\tau'_1\ldots\tau'_n)^\flat_\Sigma & = & (\tau_1\diamond^{\mathtt{T}}_1\tau'_1)^\flat_\Sigma\wedge\ldots\wedge(\tau_n\diamond^{\mathtt{T}}_n\tau'_n)^\flat_\Sigma & \\
 & & & \\
(\mathtt{a}\leqslant\tau)^\flat_\Sigma & = & (\Sigma^\uparrow(\mathtt{a})\leqslant\tau)^\flat_\Sigma & \Sigma^\uparrow(\mathtt{a})\downarrow,\ \tau\neq\mathtt{a} \\
(\alpha\leqslant\tau)^\flat_\Sigma & = & \alpha\leqslant\tau & \\
(\tau\leqslant\alpha)^\flat_\Sigma & = & \tau\leqslant\alpha & \\
 & & & \\
(\tau_1\leqslant\tau_2)^\flat_\Sigma & = & \textbf{Fail} & \text{other } \tau_1,\tau_2
\end{array}$$

We will call Π *flat* if $\Pi^\flat = \Pi$.

5.4.1 Proposition: $(\tau_1\leqslant\tau_2)^\flat$ terminates.

Proof: It suffices to demonstrate termination of $(\tau_1\leqslant\tau_2)^\flat$, which we shall do by well-foundedness of the sum of $\text{size}_\Sigma(\tau_1)+\text{size}_\Sigma(\tau_2)$. If τ_1 or τ_2 is a variable or $\tau_1=\tau_2$, the result is immediate. The case where both are constructor types results in a decomposition into inequalities of strictly smaller size. Finally suppose we are flattening $\mathtt{a} \leqslant \tau$ where τ is not a variable. If τ is $\mathtt{b}$ by a finite sequence of promotions either flattening fails or the constraint disappears. Otherwise, τ is of the form $\mathtt{T}\tau_1 \ldots \tau_n$, and $\mathtt{a}$ must eventually either promote to $\mathtt{T}\tau_1' \ldots \tau_n'$ whence we proceed as in the constructor-constructor case, or promote to a base type with no upper bound, or constructor type with a constructor other than $\mathtt{T}$, in which case flattening fails. □

Binding Let $\Pi = P\{\forall\mathtt{a}(\leqslant\tau)\Pi'\}$, where Π' is flat and contains some constraint π in which $\mathtt{a}$ occurs essentially. Since Π' is flat, there are only four possibilities for the form of π, and each of them gives rise to a substitution: If the substitution variable is existentially bound outside Π', there is no solution to the problem.

π	substitution
$\beta \leqslant \mathtt{T}\tau_1 \ldots \tau_n$	$[\mathtt{T}\beta_1 \ldots \beta_n/\beta]$
$\mathtt{T}\tau_1 \ldots \tau_n \leqslant \beta$	$[\mathtt{T}\beta_1 \ldots \beta_n/\beta]$
$\beta \leqslant \mathtt{a}$	$[\mathtt{a}/\beta]$
$\mathtt{a} \leqslant \beta$	$[\mathtt{a}/\beta]$

where all the β_i are fresh.

Projection Let $\Pi = P\{\forall\alpha(\leqslant\tau)\Pi'\}$, where Π' is flat and contains no constraint π in which $\mathtt{a}$ occurs essentially. The transition is as follows:

Old	New
$P\{\forall\mathtt{a}.\Pi'\}$	$P\{\Pi'\}$
$P\{\forall\mathtt{a}\leqslant\tau.\Pi'\}$	$\{[\tau/\mathtt{a}]\Pi'\}$

We demonstrate the use of these transitions on the problem seen earlier:

$$\begin{array}{lll}
 & & \forall\mathtt{Int}\,\forall\mathtt{Nat}\leqslant\mathtt{Int}\,\exists\gamma\,\forall\mathtt{a}\leqslant\mathtt{Nat}\,\exists\alpha.\,\gamma\to\alpha\ \leqslant\ \mathtt{a}\to\mathtt{Int}\to\mathtt{a}\\
\text{flatten} & \Rightarrow & \forall\mathtt{Int}\,\forall\mathtt{Nat}\leqslant\mathtt{Int}\,\exists\gamma\,\forall\mathtt{a}\leqslant\mathtt{Nat}\,\exists\alpha.\,\mathtt{a}\leqslant\gamma\ \wedge\ \alpha\leqslant\mathtt{Int}\to\mathtt{a}\\
\text{bind } [\alpha_1\to\alpha_2/\alpha] & \Rightarrow & \forall\mathtt{Int}\,\forall\mathtt{Nat}\leqslant\mathtt{Int}\,\exists\gamma\,\forall\mathtt{a}\leqslant\mathtt{Nat}\,\exists\alpha_1,\alpha_2.\,\mathtt{a}\leqslant\gamma\\
 & & \quad\wedge\ \alpha_1\to\alpha_2\leqslant\mathtt{Int}\to\mathtt{a}\\
\text{flatten} & \Rightarrow & \forall\mathtt{Int}\,\forall\mathtt{Nat}\leqslant\mathtt{Int}\,\exists\gamma\,\forall\mathtt{a}\leqslant\mathtt{Nat}\,\exists\alpha_1,\alpha_2.\,\mathtt{a}\leqslant\gamma\\
 & & \quad\wedge\ \mathtt{Int}\leqslant\alpha_1\ \wedge\ \alpha_2\leqslant\mathtt{a}\\
\text{bind } [\mathtt{a}/\alpha_2] & \Rightarrow & \forall\mathtt{Int}\,\forall\mathtt{Nat}\leqslant\mathtt{Int}\,\exists\gamma\,\forall\mathtt{a}\leqslant\mathtt{Nat}\,\exists\alpha_1.\,\mathtt{a}\leqslant\gamma\\
 & & \quad\wedge\ \mathtt{Int}\leqslant\alpha_1\ \wedge\ \mathtt{a}\leqslant\mathtt{a}\\
\text{flatten} & \Rightarrow & \forall\mathtt{Int}\,\forall\mathtt{Nat}\leqslant\mathtt{Int}\,\exists\gamma\,\forall\mathtt{a}\leqslant\mathtt{Nat}\,\exists\alpha_1.\,\mathtt{a}\leqslant\gamma\ \wedge\ \mathtt{Int}\leqslant\alpha_1\\
\text{project } \mathtt{a} & \Rightarrow & \forall\mathtt{Int}\,\forall\mathtt{Nat}\leqslant\mathtt{Int}\,\exists\gamma,\alpha_1.\,\mathtt{Nat}\leqslant\gamma\ \wedge\ \mathtt{Int}\leqslant\alpha_1\\
\text{project } \mathtt{Nat} & \Rightarrow & \forall\mathtt{Int}\,\exists\gamma,\alpha_1.\,\mathtt{Int}\leqslant\gamma\ \wedge\ \mathtt{Int}\leqslant\alpha_1
\end{array}$$

5.5 Soundness and Completeness

In order to demonstrate that we can decide this subtyping relation, we will demonstrate that the transitions form a sound and complete system for solving QSPs.

5.5.1 Definition: A QSP is *trivial* if there are no constrained quantifiers

If all subtyping is generated from Σ, then all trivial QSPs can be solved directly by unification. We can generalise this condition to the case, for example, where Σ extends some underlying Helly poset of atomic types; once the QSP is reduced to triviality, we can use the usual mechanisms for solvability and solution over Helly posets.

Soundness and completeness mean that given a problem Π, a finite sequence of transitions will completely reduce the QSP to a trivial QSP if and only if the problem has a solution. The definitions and results given here will enable us to prove a stronger property of *factorisation* in the next chapter.

The property that we need for this is: given a problem $P\{\Pi_1\}$ which is transformed to $P\{\Pi_2\}$ by some sequence of transitions, for any solution θ_1 of $P\{\Pi_1\}$ there will be a corresponding solution θ_2 of $P\{\Pi_2\}$ which agrees with θ_1 on the variables existentially quantified in P outside of Π_1.

We will demonstrate this strong form of completeness by considering each of the transitions in turn.

Flattening

5.5.2 Lemma: Suppose Π is a subtyping problem

1. if Π has a solution, then flattening succeeds on Π.

2. θ is a solution of $\Pi^\flat$ iff it is a solution of Π.

Proof:

1. The cases for $\tau_1 \leqslant \tau_2$ in which flattening can fail are exactly those where neither of τ_1 and τ_2 is an existentially quantified variable, and either they have different shapes, or they are both Σ-atoms such that τ_2 cannot be obtained by a sequence of promotions from τ_1. Thus there can be no solution to Π.

2. Proofs of the sequents in $S(\theta\Pi)$ can be obtained from those $S(\theta(\Pi^\flat))$ just by use of the inference rules, and similarly if θ is a solution to Π, a canonical proof of a sequent in $S(\Pi)$ must (by consideration of shape) end with the inference rule used in flattening that sequent. □

Binding

5.5.3 Lemma: Suppose $P\{\Pi_1\}$ is transformed to $P\{\Pi_2\}$ by a binding transition using a substitution $[\tau/\beta]$. Let V be the set of variables other than β which are existentially bound in $P\{\Pi_1\}$.

1. if θ_1 is a solution of $P\{\Pi_1\}$, then there is some solution θ_2 of $P\{\Pi_2\}$ such that $\theta_1|_V = \theta_2|_V$

2. if θ_2 is a solution of $P\{\Pi_2\}$, there is a solution θ_1 of $P\{\Pi_1\}$ such that $\theta_1|_V = \theta_2|_V$

Proof:

1. Suppose the binding transition arises from an essential occurrence of $\mathtt{a}$ in a constraint of the form $\beta \leqslant \mathtt{T}\tau_1 \ldots \tau_n$, and θ_1 is a solution of $P\{\Pi_1\}$. In this case $\Sigma, \mathtt{a}(\leqslant\tau_0) \;\vdash\; \theta_1\beta \leqslant \mathtt{T}(\theta_1\tau_1)\ldots(\theta_1\tau_n)$, and since $\mathtt{a}$ occurs essentially in $\mathtt{T}(\theta_1\tau_1)\rightarrow(\theta_1\tau_n)$ we know from lemma(5.2.9(2)) that $\theta_1\beta$ must be $\mathtt{T}\tau'_1 \ldots \tau'_n$. Now τ is of the form $\mathtt{T}\beta_1 \ldots \beta_n$ for fresh variables β_i. So $\theta_1|_V \circ [\tau'_i/\beta_i]$ is a solution of $P\{[\tau/\beta]\Pi_1\}$.

2. By similar reasoning, if θ_2 is a solution of $P\{[\mathtt{T}\beta_1 \ldots \beta_n/\beta]\Pi_1\}$, and the substitution comes from a binding transition, then $\theta_2|_V \circ [\theta_2(\mathtt{T}\beta_1 \ldots \beta_n)/\beta]$ is a solution to $P\{\Pi_1\}$.

The cases for the form of constraint $\beta \leqslant \mathtt{a}$ are straightforward. □

Projection

We need the following preliminary lemma:

5.5.4 Lemma: Suppose Σ is well-formed. Let $\mathtt{a}$ be a leaf in Σ, $\Sigma^{\uparrow}(\mathtt{a}) = \tau'$

1. if $\mathtt{a}$ occurs only negatively in τ, $Q, \Sigma \;\vdash\; \tau \leqslant [\tau'/\mathtt{a}]\tau$

2. if $\mathtt{a}$ occurs only positively in τ, $Q, \Sigma \;\vdash\; [\tau'/\mathtt{a}]\tau \leqslant \tau$.

Proof: We proceed by simultaneous induction on the structure of τ. Suppose τ is a type constant, $\mathtt{b}$. If $\mathtt{a} \neq \mathtt{b}$, the result is obvious. And if $\mathtt{a} = \mathtt{b}$, we must be in case (1), and a proof of $\Sigma, \mathtt{a}{\leqslant}\tau' \;\vdash\; \mathtt{a} \leqslant \tau'$ is immediate by use of Prom$^{\uparrow}$. Otherwise τ is a constructor type, and thus so is $[\tau'/\mathtt{a}]\tau$; the result follows by decomposition by Cons and the induction hypothesis. □

5.5.5 Lemma: Suppose $P\{\Pi_1\}$ is transformed to $P\{\Pi_2\}$ by a projection transition.

1. Let V be the set of variables existentially bound in the context P. If θ_1 is a solution of $P\{\Pi_1\}$, then there is some solution θ_2 of $P\{\Pi_2\}$ such that $\theta_1|_V = \theta_2|_V$

2. if θ_2 is a solution of $P\{\Pi_2\}$, there θ_2 is a solution of $P\{\Pi_1\}$.

Proof:

1. Suppose θ_1 is a solution to $P\{\forall \mathsf{a}.\Pi\}$, it suffices to show that $[\tau/\mathsf{a}] \circ \theta_1$ is a solution to $P\{\Pi\}$ for any $\tau \in \tau(\Sigma_P)$. This follows by induction on the derivation of any sequent in $S(\theta_1(P\{\forall \mathsf{a}.\Pi\}))$: every rule containing a is either Cons, Prom$^{\uparrow}$ on a base type other than a, or Refl on a.

   In the bounded case, where θ_1 is a solution to $P\{\forall \mathsf{a} \leqslant \tau.\Pi\}$ and a does not occur essentially in Π, $[\tau/\mathsf{a}] \circ \theta_1$ is again a solution to $P\{[\tau/\mathsf{a}]\Pi\}$: again this follows by induction on the derivation.

   In either case, the new substitution must agree with θ_1 on the existentially quantified variables in P, since these can contain no occurrences of a.

2. The unbounded case is simple: Suppose θ_2 is a solution to $P\{\Pi\}$. Then by weakening, it's also a solution to $P\{\forall \mathsf{a}.\Pi\}$.

   The bounded case is slightly more difficult: suppose θ_2 is a solution to $P\{[\tau/\mathsf{a}]\Pi\}$, and we have a sequent $\theta_2\Sigma_P \ \vdash \ \theta_2([\tau/\mathsf{a}]\tau_1) \leqslant \theta_2([\tau/\mathsf{a}]\tau_2)$ corresponding to some inequality $\tau_1 \leqslant \tau_2$ in Π. We know that

$$\theta_2\Sigma_P, \mathsf{a} \leqslant \theta_2\tau \ \vdash \ \theta_2([\tau/\mathsf{a}]\tau_1) \leqslant \theta_2([\tau/\mathsf{a}]\tau_2)$$

   Now since $\theta_2(\alpha)$ does not contain a for any α, $\theta_2 \circ [\tau/\mathsf{a}] = [\theta_2\tau/\mathsf{a}] \circ \theta_2$, so by rearranging substitutions we have:

$$\theta_2\Sigma_P, \mathsf{a} \leqslant \theta_2\tau \ \vdash \ [\theta_2\tau/\mathsf{a}](\theta_2\tau_1) \leqslant [\theta_2\tau/\mathsf{a}](\theta_2\tau_2)$$

   Now from lemma(5.5.4) we get

$$\theta_2\Sigma_P, \mathsf{a} \leqslant \theta_2\tau \ \vdash \ \theta_2\tau_1 \leqslant [\theta_2\tau/\mathsf{a}](\theta_2\tau_1)$$

   and

$$\theta_2\Sigma_P, \mathsf{a} \leqslant \theta_2\tau \ \vdash \ [\theta_2\tau/\mathsf{a}](\theta_2\tau_2) \leqslant \theta_2\tau_2$$

   Two uses of Trans yield the required proof. □

5.5.6 Theorem: Suppose $P\{\Pi_1\}$ is transformed to $P\{\Pi_2\}$ by a sequence of transitions. Let V be the set of variables existentially bound in P.

1. if θ_1 is a solution of $P\{\Pi_1\}$, there is a solution θ_2 of $P\{\Pi_2\}$ such that $\theta_1|_V = \theta_2|_V$.
2. if θ_2 is a solution of $P\{\Pi_2\}$, there is a solution θ_1 of $P\{\Pi_1\}$ such that $\theta_1|_V = \theta_2|_V$.

Proof: By induction on the length of the transition sequence, by cases on the last transition in the sequence. Suppose $P\{\Pi_1\}$ is transformed to $P\{\Pi_2\}$, such that the results hold for $P\{\Pi_2\}$, and $P\{\Pi_2\}$ undergoes a transition to $P\{\Pi_3\}$. We will proceed by cases on the type of transition.

flattening θ_2 is a solution to $P\{\Pi_3\}$ iff it is a solution to $P\{\Pi_2\}$.

binding Suppose the binding is a substitution θ'.

1. Note that the variable being substituted cannot be in V. Thus by lemma(5.5.3(1)), there's a solution θ_3 of $P\{\Pi_3\}$ such that $\theta_3|_V = \theta_2|_V$. The result follows from the induction hypothesis.
2. Similarly, using lemma(5.5.3(2))

projection Similarly, using lemma(5.5.5). □

5.6 Termination

So we see that transition sequences are sound and complete: it remains to demonstrate how a sequence of transitions may be applied to solve a QSP. Our strategy will be based on a step we will call *elimination* (because after the transitions which make up each such step, a universal quantifier will have been eliminated from the problem). To eliminate an innermost universal quantifier from $P\{\forall \mathtt{a}(\leqslant\tau).\Pi\}$ we flatten Π, eliminate all essential occurrences of $\mathtt{a}$ using alternating binding and flattening transitions, then remove the binder with a projection transition.

It suffices to show that no infinite sequence of flattening and binding transitions can occur. We will say Π is *shape consistent* if the corresponding mixed-prefix unification problem on shapes has some solution θ_S. We will demonstrate that if Π is shape consistent, no such circularities will occur. Obviously if Π is not shape consistent, then it is not solvable.

5.6.1 Proposition: If C is shape consistent, there can be no infinite sequence of substitutions and flattenings.

Proof: If C is a set of inequalities, we define $\text{size}_\Sigma(C)$ to be the sum of sizes of the types occurring in the inequalities in C. Let θ_S be as above, and θ be a binding substitution.

If $\theta = [\mathtt{a}/\alpha]$, we must have $\mathcal{S}(\theta_S(\mathtt{a})) = \mathcal{S}(\alpha)$ for any θ_S, and we define $\theta'_S = \theta_S$. Otherwise $\theta = [\mathtt{T}\beta_1 \ldots \beta_n/\alpha]$, and so we must have $\mathcal{S}(\theta_S(\mathtt{T}\beta_1 \ldots \beta_n)) = \mathcal{S}(\alpha)$, thus we have an extension θ'_S of θ_S such that $\theta'_S(\mathtt{T}\beta_1 \ldots \beta_n) = \mathtt{T}(\theta\beta_1) \ldots (\theta\beta_n)$, and θ'_S is also a a solution of the shape unification problem.

Now $\text{size}_\Sigma(\theta_S C) = \text{size}_\Sigma(\theta'_S(\theta C)) > \text{size}_\Sigma(\theta_S(\theta(C)^\flat))$, since flattening always either removes a constraint $\mathtt{a} \leqslant \mathtt{a}$ or decomposes a constraint with Cons.

Thus each sequence of substitutions θ^i gives rise to a sequence of substitutions θ^i_S such that

$$\text{size}_\Sigma(\theta^{i+1}_S((\theta^{i+1}C)^\flat)) < \text{size}_\Sigma(\theta^i_S((\theta^i C)^\flat))$$

The result follows by well-foundedness. □

5.6.2 Proposition: An elimination step terminates, and if Π has a solution, no elimination step on Π will fail.

Proof: Since there can be no infinite sequence of binding and flattening transitions, the elimination step must terminate. By theorem(5.5.6), each transition in the elimination process preserves solvability. The only transition which can fail is a flattening transition, but by lemma(5.5.2) flattening cannot fail on a solvable QSP. □

To solve a problem, we apply elimination steps until the problem is reduced to a normal first-order unification problem, then apply standard techniques. Notice the similarity with mixed-prefix unification. There are three possible ways in which unification may fail: constructors may mismatch, we may attempt to bind an existentially quantified variable to a type containing a universally quantified variable which is not in its scope, or we may discover circularity in an inequality (or collection thereof) of the form $\beta \leqslant \tau$ with β occurring non-trivially in τ. The first of these corresponds to failure of a flattening transition, the second failure of a binding transition, and the third to an infinite sequence of binding and flattening transitions.

5.7 Transitions and Proofs

As discussed earlier, a QSP can be seen as a set of sequents, and the solution process as a decomposition of the sequents in this set using proof rules and substitutions. It is perhaps worth making explicit the connection between the solution of a QSPs and the generation of a proof.

5.7.1 Definition: Given sets of sequents S_1 and S_2, we say S_1 *builds* S_2 and write $S_1 \Vdash S_2$ if, given proofs of the sequents in S_1, we can construct proofs of the sequents in S_2 using the inference rules between types and the admissible rules Weaken and Weaken$_{\leqslant}$.

$$\frac{\Sigma \vdash \sigma_1 \leqslant \sigma_2 \qquad \mathtt{a} \text{ does not occur in } \sigma_1 \text{ or } \sigma_2}{\Sigma, \mathtt{a} \vdash \sigma_1 \leqslant \sigma_2} \quad (\textsc{Weaken})$$

$$\frac{\begin{array}{c}\Sigma \vdash \sigma_1 \leqslant \sigma_2 \qquad \mathtt{a} \text{ does not occur in } \sigma_1 \text{ or } \sigma_2 \\ \text{all base types in } \tau \text{ occur in } \Sigma\end{array}}{\Sigma, \mathtt{a}{\leqslant}\tau \vdash \sigma_1 \leqslant \sigma_2} \quad (\textsc{Weaken}_{\leqslant})$$

5.7.2 Remark: $\Vdash$ is transitive.

5.7.3 Proposition: Suppose $S_1 \Vdash S_2$, and θ is a substitution such that all the sequents in θS_2 are well-formed. Then all the sequents in θS_1 are well-formed, and $\theta S_1 \Vdash \theta S_2$.

Proof: By induction on the proof rules. □

5.7.4 Lemma: Suppose $P\{\Pi_1\}$ is transformed to $P\{\Pi_2\}$ by a sequence of transitions. Then there is a substitution θ on the existential variables of Π_1 such that $S(P\{\Pi_1\}) \Vdash S(P(\theta\Pi_1))$.

Proof: The proof is by induction on the transition sequence. Suppose we have a substitution θ such that $S(\Pi_2) \Vdash S(\theta\Pi_1)$, and Π_2 is transformed to Π_3 by a transition. We proceed by cases:

flattening $S(\Pi_3) \Vdash S(\Pi_2)$, so the result is obvious.

binding Suppose the binding transition is the substitution θ'. Then $S(\Pi_3) \Vdash S(\theta'\Pi_2)$, and thus $S(\Pi_3) \Vdash S((\theta' \circ \theta)\Pi_1)$.

projection The unbounded case is a straightforward: uses of weakening suffice to prove that $S_3 \Vdash S_2$. In the bounded case, for any sequent in S_2 $\Sigma, \mathtt{a}{\leqslant}\tau \vdash \tau_1 \leqslant \tau_2$ we can construct a proof from $\Sigma \vdash [\tau/\mathtt{a}]\tau_1 \leqslant [\tau/\mathtt{a}]\tau_2$ in a similar fashion to lemma (5.5.5)(ii)

In the latter two cases, only variables inside Π_1 are affected by the substitution, so θ is always restricted to the variables in Π_1. □

5.8 An Algorithm for Subtyping

Having developed the theory of quantified subtyping problems, it remains to show how a subtyping query may be transformed into one. We will define two transformations, one for the signature and one for the problem, and compose the results to obtain the QSP.

The prefix $\mathrm{C}(\Sigma)$ corresponding to a signature Σ is

$$\begin{array}{lcll}
\mathrm{C}(\bullet) & = & \bullet & \text{(The empty prefix)} \\
\mathrm{C}(\Sigma, \mathtt{a}) & = & \mathrm{C}(\Sigma)\forall \mathtt{a} & \\
\mathrm{C}(\Sigma, \mathtt{a} \leqslant \tau) & = & \mathrm{C}(\Sigma)\forall \mathtt{a} \leqslant \tau &
\end{array}$$

The problem $\mathrm{I}(\sigma_1 \leqslant \sigma_2)$ corresponding to the inequality $\sigma_1 \leqslant \sigma_2$ is

$$\begin{array}{lcll}
\mathrm{I}(\sigma_1 \leqslant \forall\alpha.\sigma_2) & = & \forall \mathtt{a}.\mathrm{I}(\sigma_1 \leqslant [\mathtt{a}/\alpha]\sigma_2) & \mathtt{a}\text{ fresh} \\
\mathrm{I}(\sigma_1 \leqslant \forall\alpha\leqslant\tau.\sigma_2) & = & \forall \mathtt{a}\leqslant\tau.\mathrm{I}(\sigma_1 \leqslant [\mathtt{a}/\alpha]\sigma_2) & \mathtt{a}\text{ fresh} \\
\mathrm{I}(\forall\alpha.\sigma_1 \leqslant \sigma_2) & = & \exists\alpha.\mathrm{I}(\sigma_1 \leqslant \sigma_2) & \alpha \notin \mathrm{fv}(\sigma_2),\ \sigma_2\text{ not} \\
 & & & \text{outermost quantified} \\
\mathrm{I}(\forall\alpha\leqslant\tau.\sigma_1 \leqslant \sigma_2) & = & \exists\alpha.\alpha \leqslant \tau \wedge \mathrm{I}(\sigma_1, \sigma_2) & \alpha \notin \mathrm{fv}(\sigma_2),\ \sigma_2\text{ not} \\
 & & & \text{outermost quantified} \\
\mathrm{I}(\sigma_1\rightarrow\sigma_2 \leqslant \sigma_1'\rightarrow\sigma_2') & = & \mathrm{I}(\sigma_1' \leqslant \sigma_1) \wedge \mathrm{I}(\sigma_2 \leqslant \sigma_2') & \text{if one of } \sigma_1\rightarrow\sigma_2,\ \sigma_1'\rightarrow\sigma_2' \\
 & & & \text{is not a type} \\
\mathrm{I}(\alpha \leqslant \sigma_1\rightarrow\sigma_2) & = & \exists\alpha_1,\alpha_2.\alpha \leqslant \alpha_1\rightarrow\alpha_2 & \text{if } \sigma_1\rightarrow\sigma_2\text{ is not a type} \\
 & & \wedge\, \mathrm{I}(\sigma_1 \leqslant \alpha_1) \wedge \mathrm{I}(\alpha_2 \leqslant \sigma_2) & \\
\mathrm{I}(\sigma_1\rightarrow\sigma_2 \leqslant \alpha) & = & \exists\alpha_1,\alpha_2.\alpha_1\rightarrow\alpha_2 \leqslant \alpha & \sigma_1\rightarrow\sigma_2\text{ is not a type} \\
 & & \wedge\, \mathrm{I}(\sigma_1 \leqslant \alpha_1) \wedge \mathrm{I}(\alpha_2 \leqslant \sigma_2) & \\
\mathrm{I}(\tau_1 \leqslant \tau_2) & = & \tau_1 \leqslant \tau_2 & \\
\mathrm{I}(\sigma_1 \leqslant \sigma_2) & = & \mathbf{Fail} & \text{otherwise}
\end{array}$$

The problem $\mathrm{P}(\Sigma, \sigma_1, \sigma_2)$ is $\mathrm{C}(\Sigma)\mathrm{I}(\sigma_1 \leqslant \sigma_2)$.

5.8.1 Remark: $\mathrm{P}(\Sigma, \sigma_1 \leqslant \sigma_2)$ is a well-formed, closed quantified subtype problem

5.8.2 Lemma:

1. Suppose $\Sigma \vdash \sigma_1 \leqslant \sigma_2$. Then $\mathrm{P}(\Sigma, \sigma_1, \sigma_2)$ exists, and has a solution θ.

2. Suppose θ is a solution of $\mathrm{P}(\Sigma, \sigma_1, \sigma_2)$. Then $\theta(S(\mathrm{P}(\Sigma, \sigma_1, \sigma_2))) \Vdash \Sigma \vdash \sigma_1 \leqslant \sigma_2$.

Proof:

1. We use the following induction hypothesis: suppose there is a substitution θ such that $\theta\Sigma \vdash \theta\sigma_1 \leqslant \theta\sigma_2$. Then $\mathrm{P}(\theta\Sigma, \theta\sigma_1, \theta\sigma_2)$ exists, and has a solution θ' which is an extension of θ. We proceed by induction on the cases for $\mathrm{I}(\sigma_1, \sigma_2)$: the first five cases are straightforward, corresponding to the rule applications of a canonical derivation of $\theta\Sigma \vdash \theta\sigma_1 \leqslant \theta\sigma_2$. For the next, it suffices to note that if we have an instance of $\theta\alpha \leqslant \theta\sigma_1 \leqslant \theta\sigma_2$ then the last rule in the derivation must be either Arrow or $\text{Prom}^{\uparrow}$. If it is the former, then we have $\theta(\alpha) = \tau_1 \to \tau_2$, so we set $\theta'\alpha_i \leqslant \tau_i$. Otherwise there must be some series of $\text{Prom}^{\uparrow}$ rules below an arrow rule, in which case we still have $\tau_1 \to \tau_2$ such that $\theta\alpha \leqslant \tau_1 \to \tau_2$. Finally, if we have an instance of $\theta\Sigma \vdash \theta\sigma_1 \to \theta\sigma_2 \leqslant \theta(\alpha)$ and $\sigma_1 \to \sigma_2$ is not a type, then the last rule must be Arrow, so $\theta(\alpha) = \tau_1 \to \tau_2$, and so if $\theta'[\tau_1/\beta_1] \circ [\tau_2/\beta_2] \circ \theta$, we have $\theta'\Sigma \vdash \theta'(\beta_1 \to \beta_2) \leqslant \theta'\sigma_2$.

2. Similarly, if we have a solution θ to $\mathrm{I}(\Sigma, \sigma_1, \sigma_2)$, we can construct a proof of $\theta\Sigma \vdash \theta\sigma_1 \leqslant \theta\sigma_2$ □

5.9 Conclusion

In this chapter we have shown how bounded subtyping can be incorporated into a unification-style framework in order to decided a subtype relation which marries subtype and parametric polymorphism. In fact, QSPs are very similar to mixed-prefix unification problems, and we will exploit this in the next chapter to generate constraint sets which play an analogous role to principal solutions.

We have been unable to extend this approach to deal with non-structural subtyping, for example with records. The problems lies in the binding transition: if we have an inequality of the form $\alpha \leqslant \{l_i{:}\tau_i\}_{i \in I}$ where some base type in the record type occurs essentially, it is not possible in general to determine a shape for α such that the substitution will preserve all possible solutions and introduce no new ones. Although we could use the techniques of chapter 4 to find a set of shapes which we could use to solve this problem, such a solution only works in the context of a closed problem: While this still has applications (for example, signature matching), it will not be useful for applications such as the next chapter, where QSPs need to be partially solved before the shape of all the type variables they contain can be fully determined.

A further problem with the approach here is that the technique used for solution falls into the class of "string-based" rather than "graph-based" unification strategies. In theory such approaches can cause exponential increase in the size of constraint sets; but typically, bounded signatures are small and constraint sets contain few essential occurrences of the types to be eliminated, and the problem has not occurred in any reasonable example. The situation corresponds to the similar case in typed λ-calculus where the printed form of an inferred type is huge, and such cases appear not to arise in practice.

Chapter 6

Subtyping and Annotation

Like ML, $\mathrm{ML}_{\leqslant}$ lacks the ability to pass function parameters with polymorphic types. For example, consider the function

```
fun pair_app f (a,b) (c,d) = (f a c, f b d)
```

if we have $\texttt{max} : \forall\alpha\leqslant\texttt{Int}.\ \alpha\to\alpha\to\alpha$, then we cannot type the expression

```
pair_app max (3, -7) (4,-24)
```

and obtain the value `(4,-7)` of type $\texttt{Nat} \times \texttt{Int}$ since it requires `max` to be used with types $\texttt{Nat}\to\texttt{Nat}\to\texttt{Nat}$ and $\texttt{Int}\to\texttt{Int}\to\texttt{Int}$, which have no common subtype.

In an analogous fashion to the lifting of this restriction for ML in [31], here we lift that restriction for $\mathrm{ML}_{\leqslant}$, by requiring annotations on λ-bound variables whose arguments are intended to be of polymorphic type. All unannotated variables are assumed to be of monomorphic type, as they would be in ML. For example, the above term becomes

```
fun pair_app (f:All('a<'b) 'a -> 'a -> 'a) (a,b) (c,d) =
                  (f a c, f b d)
```

for which a type is inferred which has as an instance the type scheme

$$\forall\beta\,(\forall\alpha\leqslant\beta.\alpha\to\alpha\to\alpha) \to \forall\gamma\leqslant\beta\,\forall\delta\leqslant\beta.\,(\gamma\times\delta)\to(\gamma\times\delta)\to(\gamma\times\delta)$$

(As normal, the scope of universal quantifiers extend as far to the right as possible, while → is right-associative). We will use our theory of subtyping to define $\mathrm{ML}_{\forall\leqslant}$, a type system incorporating both subtyping and parametric polymorphism, and the ability to pass polymorphic functions such as `max` as parameters. Although we have defined a subtype relation incorporating first-order type schemes and subtyping, there are still issues which need to be tackled before it can be incorporated into a type inference system.

Free Variables

As in $\mathrm{ML}_{\leqslant}$, any type inference system will need to be able to check inclusions between types which contain variables not bound in the type. We will modify our subtyping algorithm to deal with these.

Completeness

The system of types as presented cannot be incorporated into an ML-like system which has complete type inference. For example, suppose we have a term f of type

$\forall\alpha.(\alpha\to\alpha\to\mathtt{Int})\to\alpha\to\alpha\to\mathtt{Int}$, and K is $\lambda x\lambda y.x$.

Consider the term

$$(\lambda z.Kz(fz))(\lambda x\lambda y.3)$$

The term $\lambda x\lambda y.3$ can be given any type $\alpha\to\beta\to\mathtt{Int}$, and the term $\lambda z.Kz(fz)$ any type $\gamma\to\gamma$ such that $\gamma\leqslant\delta\to\delta\to\mathtt{Int}$ for some δ. So combining these, the whole expression can be given any type $\alpha\to\beta\to\mathtt{Int}$ which has the property that $\alpha\to\beta\to\mathtt{Int}\leqslant\delta\to\delta\to\mathtt{Int}$ for some δ, i.e. such that there is some δ for which $\delta\leqslant\alpha$ and $\delta\leqslant\beta$. However, our type system does not let us express a type scheme which can be instantiated with any two types possessing a common lower bound. Thus if we do not widen our notion of type scheme, we lose some of the possible instantiations of inferred types.

Types and Environments

We will prove a weak version of principality for our type inference system, which we shall term *minimality*. In order to obtain minimal typing, we extend the definition of type schemes once more with *constrained type schemes*, which can be seen as a generalisation of the constrained type-schemes for $\mathrm{ML}_{\leqslant}$.

$$\kappa ::= \alpha \quad|\quad \sigma \quad|\quad \sigma\to\kappa \quad|\quad \forall\alpha.\kappa \quad|\quad \forall\alpha{\leqslant}\tau.\kappa \quad|\quad \forall\overline{\alpha}\backslash C.\kappa$$

Notice that the domain type of a function must be a *bounded* type scheme σ. And instead of having a separate signature for subtyping information, we will have a single environment incorporating term and type bindings.

$\bullet$	The empty context
$\Gamma, \mathtt{a}$	
$\Gamma, \mathtt{a}\leqslant\tau$	where Γ contains all base types in τ
$\Gamma, x:\kappa$	where Γ contains all base types in κ.

A scoping restriction will be enforced on free variables by only allowing substitutions θ such that $\theta\Gamma$ is well-formed. We will call such substitutions Γ-substitutions.

Although we shall define the notion more formally later, intuitively a type scheme κ_1 is minimal for an expression if for any other derived type κ_2 and any σ such that $\kappa_2 \leqslant \sigma$, we obtain $\kappa_1 \leqslant \sigma$. This is a very weak form of principal type in the sense of [23]: although we cannot derive all possible type schemes from a minimal type, we can derive all *bounded* type schemes. We restrict contravariant occurrences of type schemes to be bounded type schemes, and this will ensure that the modularity requirement is met that if we type a term with its minimal type, the term will be usable in any context where any different type that we might assign it would suffice. Thus our type inference algorithm will be complete.

6.1 Extensions to the Type System

In order to avoid the complexities which occur when dealing with type assertions of the form $\kappa_1 \leqslant \kappa_2$, we will permit constrained types to appear only in assertions of the form $\kappa \leqslant \sigma$. We will ensure this by placing a syntactic restriction on programs: constrained type schemes are not legal in term syntax. Then the following subtyping rule for constrained types suffices:

$$\frac{\Gamma \vdash [\overline{\tau}/\overline{\alpha}]C \qquad \Gamma \vdash [\overline{\tau}/\overline{\alpha}]\kappa \leqslant \sigma}{\Gamma \vdash \forall\overline{\alpha}\backslash C.\kappa \leqslant \sigma} \qquad (\forall_{C\leqslant}\text{-LEFT})$$

This is the only subtyping rule in which constrained types are permitted, although of course $\forall$-LEFT′ and $\forall_{\leqslant}$-LEFT′ are just specialisations of $\forall_C$-LEFT. In fact we do not even have reflexivity for constrained type schemes in the subtyping system.

Since we are now not dealing with closed types, we also extend the REFL rule to include free variables, which are simply considered to be rigid.

6.1.1 Proposition: When the rule $\forall_{C\leqslant}$-LEFT is added to the system, we retain transitivity elimination and thus the notion of canonical derivations.

Constrained Judgements

In line with ML$_{\leqslant}$, we return to a constrained form of judgement: the judgements in the type system will be of the form

$$\Gamma; C \vdash e : \kappa$$

where C is a set of inequalities $\tau_1 \leqslant \tau_2$ between types. Thus we modify our subtyping judgements rules to be of the form:

$$\Gamma; C \;\vdash\; \kappa \leqslant \sigma$$

modifying $\forall_{\leqslant}$-RIGHT and $\forall$-RIGHT so that all occurrences are captured by quantification, e.g.:

$$\frac{\Gamma, \mathsf{a}; C \;\vdash\; \sigma_1 \leqslant [\mathsf{a}/\alpha]\sigma_2 \qquad \mathsf{a} \text{ does not occur in } \sigma_1 \text{ or } C}{\Gamma; C \;\vdash\; \sigma_1 \leqslant \forall\alpha.\sigma_2} \quad (\forall\text{-RIGHT})$$

and add the rule

$$\frac{\tau_1 \leqslant \tau_2 \in C}{\Gamma; C \;\vdash\; \tau_1 \leqslant \tau_2} \quad (\text{HYP})$$

to our subtyping system. In the presence of HYP, transitivity can no longer be eliminated. However, we have the following:

6.1.2 Proposition: Suppose $\Gamma; C \;\vdash\; \kappa \leqslant \sigma$ and for some Γ-substitution θ, $\Gamma \;\vdash\; \theta C$. Then $\Gamma \;\vdash\; \theta\kappa \leqslant \theta\sigma$.

Proof: By induction on the derivation of $\Gamma; C \;\vdash\; \kappa \leqslant \sigma$, each instance of HYP being replaced by the proof of the constraint. □

6.1.3 Definition: A type scheme κ is Γ-*feasible* if there is a type scheme σ such that $\Gamma \;\vdash\; \kappa \leqslant \sigma$. An environment Γ is *feasible* if for every x bound in Γ there is some σ such that $\Gamma \;\vdash\; \Gamma(x) \leqslant \sigma$.

A sequent $\Gamma; C \;\vdash\; e : \kappa$ is feasible if there is a Γ-substitution θ such that $\theta\Gamma$ is feasible, $\Gamma \;\vdash\; \theta C$ and $\theta\kappa$ is Γ-feasible.

Like $\mathrm{ML}_{\leqslant}$, the type system is quite capable of producing an infeasible derivation, but unlike $\mathrm{ML}_{\leqslant}$, some of the constraints in the derivation may be contained within the derived type scheme at arbitrary depth. A check to ensure that some appropriate substitution for constrained variables exists will require constraints to be extracted from the type scheme and combined with those at top-level, thus we introduce the following:

6.1.4 Definition [Stripping]: We use the following auxiliary rules to define the notion of stripping.

$$\frac{}{\text{strip } \tau = (\varnothing, \tau)}$$

$$\frac{\text{strip } \kappa = (C, \sigma)}{\text{strip } \sigma'{\rightarrow}\kappa = (C, \sigma'{\rightarrow}\sigma)}$$

$$\frac{\text{strip } [\gamma/\alpha]\kappa = (C, \sigma)}{\text{strip } \forall\alpha.\kappa = (C, \sigma)}$$

$$\frac{\text{strip } [\gamma/\alpha]\kappa = (C, \sigma)}{\text{strip } \forall\alpha{\leqslant}\tau.\kappa = (\{\gamma{\leqslant}\tau\} \cup C, \sigma)\}}$$

$$\frac{\text{strip } [\overline{\gamma}/\overline{\alpha}]\kappa = (C, \sigma)}{\text{strip } \forall\alpha\backslash C'.\kappa = ([\overline{\gamma}/\overline{\alpha}]C' \cup C, \sigma)}$$

where the γ are fresh.

6.1.5 Proposition: If strip $\kappa = (C, \sigma)$, then $\Gamma; C \vdash \kappa \leqslant \sigma$.

Proof: By induction on the derivation of strip κ. □

If C is unsolvable, then κ cannot be feasible, thus we can test the feasibility of $\Gamma; C' \vdash e : \kappa$ by stripping all the types and checking solvability of the accumulated constraint set over Γ.

6.2 The Type System ML$_{\forall\leqslant}$

Now we are ready to define the ML$_{\forall\leqslant}$ type system.

$$\frac{\Gamma(x) = \kappa}{\Gamma \vdash x : \kappa} \qquad (\textsc{Taut})$$

$$\frac{\Gamma, x{:}\tau; C \vdash e : \kappa}{\Gamma; C \vdash \lambda x.e : \tau{\rightarrow}\kappa} \qquad (\textsc{Abs})$$

$$\frac{\begin{array}{c}\Gamma, \mathtt{A}, x{:}[\mathtt{a}_i/a_i]\sigma; C \vdash [\mathtt{a}_i/a_i]e : \kappa \\ \Gamma, \mathtt{A}; C \vdash \kappa \leqslant \sigma' \text{ for some } \sigma' \\ \text{no } \mathtt{a}_i \text{ occurs in } C \qquad \alpha_i \text{ fresh}\end{array}}{\Gamma; C \vdash \lambda x{:}\sigma.e : [\alpha_i/a_i]\sigma{\rightarrow}[\alpha_i/\mathtt{a}_i]\kappa} \qquad (\textsc{TypedAbs})$$

where $\text{fv}(\sigma) = a_1 \ldots a_n$ and $\mathtt{A} = \mathtt{a}_1 \ldots \mathtt{a}_n$.

$$\frac{\Gamma; C \vdash e_1 : \sigma{\rightarrow}\kappa \qquad \Gamma; C \vdash e_2 : \sigma}{\Gamma; C \vdash e_1 e_2 : \kappa} \qquad (\textsc{App})$$

$$\frac{\Gamma; C_1 \;\vdash\; e_1 : \kappa_1 \qquad \Gamma, x{:}\kappa_1; C_1 \;\vdash\; e_2 : \kappa_2 \qquad \text{strip } \kappa_1 = (C_2, \sigma_1)}{\Gamma; C_1 \cup C_2 \;\vdash\; \texttt{let } x = e_1 \texttt{ in } e_2 : \kappa_2} \quad (\textsc{Let})$$

$$\frac{\Gamma; C \;\vdash\; e_1 : \sigma \qquad \sigma \in \sigma(\Gamma)}{\Gamma; C \;\vdash\; (e_1{:}\sigma) : \sigma} \quad (\textsc{Coerce})$$

$$\frac{\Gamma; C \;\vdash\; e : \kappa \qquad \Gamma; C_2 \;\vdash\; \kappa \leqslant \sigma}{\Gamma; C \;\vdash\; e : \sigma} \quad (\textsc{Sub})$$

$$\frac{\Gamma; C_1 \cup C_2 \;\vdash\; e_1 : \kappa \qquad \overline{\alpha} \cap (\mathrm{fv}(\Gamma) \cup \mathrm{fv}_\Gamma(C_2)) = \varnothing}{\Gamma; C_1 \;\vdash\; e_1 : \forall \overline{\alpha} \backslash C_2 . \kappa} \quad (\textsc{Gen})$$

$$\frac{\Gamma; C_1 \;\vdash\; e : \forall \overline{\alpha} \backslash C_2 . \kappa}{\Gamma; C_1 \cup C_2 \;\vdash\; e : \kappa} \quad (\textsc{Inst})$$

We take the view that if a variable occurs free in a type annotation, then that variable is implicitly universally quantified, and the term should be typable for any instantiation of it – for example the term $\lambda x{:}a.\, x + 3$ is not typable. Thus we impose the restriction on TypedAbs that the term is typable with a held rigid. Once this requirement is met, we substitute free variables for the a_i.

In the Let rule, stripping the type of the bound variable is necessary to ensure that constraints are not lost by being bound with a Gen rule into a type scheme for an expression used in a `let` construction in which the bound variable does not occur in the body. Since with strict semantics the term will be evaluated even though never used, it would be unsound to let the constraints vanish in this way. This observation was first made in [46], and is considered more fully in [47]. Our approach is the one taken in [15].

6.3 Feasibility and Factorisation

In this section we will develop a notion of minimal types for $\mathrm{ML}_{\forall\leqslant}$. In ML, the foundation of the principal type property is the notion of a principal unifier, one which is more general than any other unifier. Unfortunately, QSPs do not in general have principal solutions: for example the problem mentioned earlier

$$\forall \texttt{Int} \forall \texttt{Nat} \leqslant \texttt{Int} \exists \gamma \forall \texttt{a} \leqslant \texttt{Nat} \exists \alpha . \gamma \rightarrow \alpha \leqslant \texttt{a} \rightarrow \texttt{Int} \rightarrow \texttt{a}$$

has solutions $[\texttt{Int}\rightarrow\texttt{a}/\alpha] \circ [\texttt{Int}/\gamma]$ and $[\texttt{Int}\rightarrow\texttt{a}/\alpha] \circ [\texttt{Nat}/\gamma]$. There is no value we can assign to γ to define a principal solution θ, but if we leave γ completely flexible, then $[\texttt{Nat}\rightarrow\texttt{Nat}/\gamma] \circ \theta$ would need to be a solution (which it isn't).

Working with subtyping hypothesis involving constraints offers us the possibility of deriving principal constraint sets instead of principal substitutions. We shall call these constraint sets *factorisations*.

Let us consider the reasons why principal solutions do not exist. In unification problems a variable is only ever instantiated to a value to which it is known to be necessarily equal, and thus in the instantiation no solution is lost. However, in solving QSPs, while this holds for binding transitions, and flattening does not affect solutions, a projection transition replaces a base type with its bound, which preserves solvability (in the context in which it is applied) but may lose solutions.

For example, in applying the transitions to solve the QSP, we lose a possible solution: on page 104 when we projected Nat, the solution $[\mathtt{Nat}/\gamma] \circ [\mathtt{Int}{\rightarrow}\mathtt{a}/\alpha]$ of the original problem was no longer a solution of the new one. However, a solution was preserved which is identical to this one on the variables quantified outside Nat (a set which is in this case empty).

Quantified subtype problems generated during type inference will contain two types of variables, which we may call internal and external, with much the same connotations as in chapter 3. We will derive factorisations by eliminating exactly those universally bound variables generated during the reduction of $\kappa \leqslant \sigma$ to a QSP. Thus only internal variables will ever be instantiated by transitions, and if two solutions coincide on their external variables, we will retain at least one. We need a couple of technical results:

6.3.1 Proposition: If $\Gamma; C \;\vdash\; \kappa \leqslant \sigma$, $\mathtt{a}$ does not appear in this sequent, and $\tau \in \tau(\Gamma)$, then $\Gamma, \mathtt{a}({\leqslant}\tau); C \;\vdash\; \kappa \leqslant \sigma$.

Proof: By the standard method for weakening. An auxiliary renaming lemma is required, similar to that of lemma(5.1.6), whose proof is straightforward. □

6.3.2 Proposition: Suppose we have S and Γ such that any sequent in S is of the form $\Gamma \;\vdash\; \tau_1 \leqslant \tau_2$, and $S \Vdash \{\Gamma \;\vdash\; \kappa \leqslant \sigma\}$. Then $\Gamma; C \;\vdash\; \kappa \leqslant \sigma$, where C is the set of constraints given by the types $\tau_1 \leqslant \tau_2$ in S.

Proof: By induction on the construction of the proof of $\Gamma \;\vdash\; \kappa \leqslant \sigma$ from the sequents in S. □

Feasibility

Before considering the question of factorisations, we will deal with the simpler issue of feasibility; that is, the question of when a type or judgement "makes sense". This is equivalent to the question of whether a constraint set C obtained

by stripping is solvable. Evidently this is very similar to the application at the end of the last chapter; however it may be the case that the environment Γ has term bindings which contains variables occurring in C, and we need to respect the scoping restrictions this places on possible solutions.
The prefix $\mathrm{C}(\Gamma)$ corresponding to Γ is

$$\begin{array}{lcll}
\mathrm{C}(\bullet) & = & \bullet & \text{(The empty prefix)} \\
\mathrm{C}(\Gamma, \mathtt{a}) & = & \mathrm{C}(\Gamma)\forall\mathtt{a} & \\
\mathrm{C}(\Gamma, \mathtt{a}{\leqslant}\tau) & = & \mathrm{C}(\Gamma)\forall\mathtt{a}{\leqslant}\tau & \\
\mathrm{C}(\Gamma, x{:}\tau) & = & \mathrm{C}(\Gamma)\exists\overline{\beta} & \text{where } \overline{\beta} \text{ is the set of variables in } \tau \\
 & & & \ \text{not quantified in } \mathrm{C}(\Gamma)
\end{array}$$

6.3.3 Proposition: Let $\overline{\beta}$ be the set of variables occurring in C but not in Γ. Then there is a Γ-substitution θ such that $\Gamma \ \vdash \ \theta C$ iff there is a solution to the QSP

$$\mathrm{C}(\Gamma)\exists\overline{\beta}.\bigwedge C$$

Proof: By the definition of solution for QSPs. □

Factorisation

The problem $\mathrm{I}(\kappa \leqslant \sigma)$ corresponding to the inequality $\kappa \leqslant \sigma$ is

$$\begin{array}{lcll}
\mathrm{I}(\kappa \leqslant \forall\alpha.\sigma) & = & \forall\mathtt{a}.\mathrm{I}(\kappa \leqslant [\mathtt{a}/\alpha]\sigma) & \mathtt{a} \text{ fresh} \\
\mathrm{I}(\kappa \leqslant \forall\alpha{\leqslant}\tau.\sigma) & = & \forall\mathtt{a}{\leqslant}\tau.\mathrm{I}(\kappa \leqslant [\mathtt{a}/\alpha]\sigma) & \mathtt{a} \text{ fresh} \\
\mathrm{I}(\forall\alpha.\kappa \leqslant \rho) & = & \exists\alpha.\mathrm{I}(\kappa \leqslant \sigma) & \rho \text{ not quantified, } \alpha \notin \mathrm{fv}(\rho) \\
\mathrm{I}(\forall\alpha{\leqslant}\tau.\kappa \leqslant \rho) & = & \exists\alpha.\alpha \leqslant \tau \wedge \mathrm{I}(\kappa, \rho) & \rho \text{ not quantified, } \alpha \notin \mathrm{fv}(\rho) \\
\mathrm{I}(\forall\overline{\alpha}\backslash C.\kappa \leqslant \rho) & = & \exists\overline{\alpha}.C \wedge \mathrm{I}(\kappa, \rho) & \rho \text{ not quantified, } \alpha \notin \mathrm{fv}(\rho) \\
\mathrm{I}(\sigma{\rightarrow}\kappa \leqslant \sigma_1{\rightarrow}\sigma_2) & = & \mathrm{I}(\sigma_1 \leqslant \sigma) \wedge \mathrm{I}(\kappa \leqslant \sigma_2) & \text{if one of } \sigma,\ \kappa,\ \sigma_1,\ \sigma_2 \text{ is} \\
 & & & \text{not a type} \\
\mathrm{I}(\alpha \leqslant \sigma_1{\rightarrow}\sigma_2) & = & \exists\alpha_1, \alpha_2.\alpha \leqslant \alpha_1{\rightarrow}\alpha_2 & \text{if one of } \sigma_1,\ \sigma_2 \text{ is} \\
 & & \wedge\, \mathrm{I}(\sigma_1 \leqslant \alpha_1) \wedge \mathrm{I}(\alpha_2 \leqslant \sigma_2) & \text{not a type} \\
\mathrm{I}(\sigma{\rightarrow}\kappa \leqslant \alpha) & = & \exists\alpha_1, \alpha_2.\alpha_1{\rightarrow}\alpha_2 \leqslant \alpha & \text{if one of } \sigma,\ \kappa \text{ is not a type} \\
 & & \wedge\, \mathrm{I}(\alpha_1 \leqslant \sigma) \wedge \mathrm{I}(\kappa \leqslant \alpha_2) & \\
\mathrm{I}(\tau_1 \leqslant \tau_2) & = & \tau_1 \leqslant \tau_2 & \\
\mathrm{I}(\kappa \leqslant \sigma) & = & \textbf{Fail} & \text{otherwise}
\end{array}$$

Let $\overline{\alpha}$ be the set of free variables in $\mathrm{I}(\kappa \leqslant \sigma)$ not existentially quantified in $\mathrm{C}(\Gamma)$. Then the problem $\mathrm{P}(\Gamma, \kappa, \sigma)$ is $\mathrm{C}(\Gamma)\exists\overline{\alpha}.\mathrm{I}(\kappa \leqslant \sigma)$,

6.3.4 Remark: $\mathrm{P}(\Gamma, \kappa, \sigma)$ is a well-formed, closed QSP.

6.3.5 Definition [Factorisation]: A *factorisation* for Γ, κ, σ is a constraint set C such that

- if $\Gamma; C \vdash \kappa \leqslant \sigma$.
- if $\Gamma \vdash \theta\kappa \leqslant \theta\sigma$ then there is an extension θ' of θ such that $\Gamma \vdash \theta' C$.

Thus a factorisation for Γ, κ, and σ characterises all possible Γ-substitutions θ such that $\Gamma \vdash \theta\kappa \leqslant \theta\sigma$. Note that by proposition(6.1.2) for any solution θ of C we have $\Gamma \vdash \theta\kappa \leqslant \theta\sigma$.

The procedure *factor*(Γ, κ, σ) is as follows: we apply elimination steps to $\mathrm{P}(\Gamma, \kappa, \sigma)$ until we reach a QSP of the form

$$\mathrm{C}(\Gamma)\exists\overline{\alpha}'.C$$

where C is of the form $\tau_1 \leqslant \tau_1' \wedge \ldots \wedge \tau_n \leqslant \tau_n'$. Then we returns C.

Properties of Factorisation

6.3.6 Lemma:

1. Suppose there is a Γ-substitution θ such that $\Gamma \vdash \theta\kappa \leqslant \theta\sigma$. Then $\mathrm{P}(\Gamma, \kappa, \sigma)$ exists, and there is θ', an extension of θ on the variables existentially quantified in $\mathrm{I}(\kappa, \sigma)$ such that θ' is a solution of $\mathrm{I}(\Gamma, \kappa, \sigma)$.
2. Suppose θ is a solution of $\mathrm{P}(\Gamma, \kappa, \sigma)$. Then $S(\theta\mathrm{P}(\Gamma, \kappa, \sigma)) \Vdash \{\Gamma \vdash \kappa \leqslant \sigma\}$.

Proof: Essentially the same as lemma(5.8.2) □

6.3.7 Proposition: Suppose $\Gamma \vdash \theta_s\kappa \leqslant \theta_s\sigma$ for some θ well-scoped on Γ.Then

1. *factor*(Γ, κ, σ) succeeds, with C, say.
2. $\Gamma; C \vdash \kappa \leqslant \sigma$ (so for any solution θ of C we have $\Gamma \vdash \theta\kappa \leqslant \theta\sigma$)
3. There is an extension θ_s' of θ_s by the new free variables contained in C such that $\Gamma \vdash \theta_s' C$

Proof:

1. From lemma 6.3.6(i), we know that $\mathrm{P}(\Gamma, \kappa, \sigma)$ exists, and has a solution. So by proposition(5.6.2), a sequence of elimination steps on $\mathrm{I}(\Gamma, \kappa, \sigma)$ succeeds.
2. Suppose $\Pi = \mathrm{P}(\Gamma, \kappa, \sigma)$, and a sequence of eliminations in *factor* yields Π'. We know by lemma(5.7.4) that there is a substitution on the variables existentially quantified in $\mathrm{I}(\kappa, \sigma)$ such that $S(\Pi') \Vdash S(\theta\Pi)$. And $S(\theta\Pi) \Vdash \{\Gamma \vdash \kappa \leqslant \sigma\}$ by proposition(5.6.2). Thus we have $S(\Pi') \Vdash \{\Gamma \vdash \kappa \leqslant \sigma\}$, and the result follows by lemma(6.3.2).

3. By lemma(6.3.6) we have a solution θ of $\mathrm{P}(\Gamma, \kappa, \sigma)$ which agrees with θ_s on the variables quantified outside $\mathrm{I}(\kappa, \sigma)$ (which includes all the variables in C); so by theorem(5.5.6) there is a solution of $\mathrm{C}(\Gamma)\exists\overline{\alpha}'.C$ which agrees with θ_s on these variables. □

Thus $\mathit{factor}(\Gamma, \kappa, \sigma)$ yields a factorisation for Γ, κ, and σ if one exists.

6.4 Type Inference

Our approach to type inference will concentrate on obtaining minimal types rather than principal typings. Thus we concentrate on the solutions of constraint sets and instances of constrained types rather than on syntactic entailment between them. The advantage of this approach is that we obtain a relatively straightforward proof of minimality which suffices to demonstrate that inference is compositional and thus complete. The disadvantage is that we do not thereby obtain a notion of principal typing, nor a notion of equivalence on judgements which would be required in order to formalise a notion of simplification within the type system.

To obtain a type inference algorithm we adopt the usual practice of absorbing the Gen, Inst, and Sub rules into the others to get a syntax-directed algorithm.

6.4.1 Definition [Opening and Closing]: Given an environment Γ, a set of constraints C, and a type scheme ξ, we define $\mathrm{close}(\Gamma, C, \xi)$ to be $\forall\overline{\alpha}\backslash C.\xi$, where $\overline{\alpha}$ is the set of variables occurring in C or ξ but not in Γ.

In a similar vein to stripping, we recursively define the operation open on a type scheme as follows.

$$\frac{\text{open } [\beta/\alpha]\kappa = (\overline{\gamma}, C_1, \kappa_1)}{\text{open } \forall\alpha.\kappa = (\{\beta\} \cup \overline{\gamma}, C_1, \kappa_1)}$$

$$\frac{\text{open } [\beta/\alpha]\kappa = (\overline{\gamma}, C_1, \kappa_1)}{\forall\alpha{\leqslant}\tau.\kappa = (\{\beta\} \cup \overline{\gamma}, \{\beta \leqslant \tau\} \cup C_1, \kappa_1)}$$

$$\frac{\text{open } [\overline{\beta}/\overline{\alpha}]\kappa = (\overline{\gamma}, C_1, \kappa_1)}{\forall\overline{\alpha}\backslash C.\kappa = (\overline{\beta} \cup \overline{\gamma}, [\overline{\beta}/\overline{\alpha}]C \cup C_1, \kappa_1)}$$

$$\frac{\kappa \text{ is not quantified at outermost level}}{\text{open } \kappa = (\varnothing, \varnothing, \kappa)}$$

Informally, opening a type scheme κ extracts the minimal type scheme which is an instance of κ and which is not quantified at outermost level, analogous to instantiating a type scheme to a type in ML. The following proposition expresses this intuition.

6.4.2 Proposition: If $\Gamma \vdash \theta\kappa \leqslant \sigma$, σ is not quantified at outermost level, and open $\kappa = (\overline{\alpha}, C, \kappa')$, then there is an extension θ' of θ on the variables $\overline{\alpha}$ such that $\Gamma \vdash \theta'\kappa' \leqslant \sigma$.

Proof: By induction on the calculation of open κ. Note that each rule in the derivation corresponds to some $\forall$-LEFT$'$, $\forall_{\leqslant}$-LEFT$'$, or $\forall_C$-LEFT rule in the canonical derivation of $\Gamma \vdash \theta\kappa \leqslant \sigma$, from which we can build the necessary extension of θ. □

An Algorithmic Type Inference System for ML$_{\forall\leqslant}$

As with ML$_{\leqslant}$, we incorporate the SUB, GEN, and INST rules into the others to leave a single rule for each syntactic term construct.

$$\frac{\Gamma(x) = \xi}{\Gamma \vdash_A x : \xi} \qquad (\text{A:}\textsc{Taut})$$

$$\frac{\Gamma, x{:}\alpha \vdash_A e : \xi}{\Gamma \vdash_A \lambda x.e : \forall\alpha.\alpha{\to}\xi} \qquad (\text{A:}\textsc{Abs})$$

$$\frac{\Gamma, \mathtt{A}, x{:}[\mathtt{a}_i/a_i]\sigma \vdash_A e : \xi \qquad \xi \text{ is } \Gamma, \mathtt{A}\text{-feasible}}{\Gamma \vdash_A \lambda x{:}\sigma.e : \forall\alpha_i.[\alpha_i/a_i]\sigma{\to}[\alpha_i/\mathtt{a}_i]\xi} \qquad (\text{A:}\textsc{TypedAbs})$$

where $\mathrm{fv}(\sigma) = a_1 \ldots a_n$ and $\mathtt{A} = \mathtt{a}_1 \ldots \mathtt{a}_n$.

$$\frac{\begin{array}{c}\Gamma \vdash_A e_1 : \xi_1 \qquad \Gamma \vdash_A e_2 : \xi_2 \\ \text{open } \xi_1 = (\overline{\alpha}, C_1, \sigma{\to}\xi_1') \\ \mathit{factor}(\Gamma, \xi_2, \sigma) = C_2\end{array}}{\Gamma \vdash_A e_1 e_2 : \mathrm{close}(\Gamma, C_1 \cup C_2, \xi_1')} \qquad (\text{A:}\textsc{AppArrow})$$

$$\frac{\begin{array}{c}\Gamma \vdash_A e_1 : \xi_1 \;\; \Gamma \vdash_A e_2 : \xi_2 \;\; \text{open } \xi_1 = (\overline{\beta}, C_1, \alpha) \\ \mathit{factor}(\Gamma, \xi_2, \alpha_1) = C_2 \qquad \alpha_1, \alpha_2 \text{ fresh}\end{array}}{\Gamma \vdash_A e_1 e_2 : \mathrm{close}(\Gamma, C_1 \cup C_2 \cup \{\alpha \leqslant \alpha_1{\to}\alpha_2\}, \alpha_2)} \qquad (\text{A:}\textsc{AppVar})$$

$$\frac{\Gamma \vdash_A e_1 : \xi_1 \qquad \Gamma, x : \xi_1 \vdash_A e_2 : \xi_2 \qquad \text{strip } \xi_1 = (C, \sigma_1)}{\Gamma \vdash_A \mathtt{let}\ x = e_1\ \mathtt{in}\ e_2 : \mathrm{close}(\Gamma, C, \xi_2)} \qquad (\text{A:}\textsc{Let})$$

$$\frac{\Gamma \vdash_A\ e_1{:}\xi \qquad \mathit{factor}(\Gamma, \xi, \sigma) = C \qquad \sigma \in \sigma(\Gamma)}{\Gamma \vdash_A\ (e_1{:}\sigma) : \mathrm{close}(\Gamma, C, \sigma)} \quad (\text{A:}\textsc{Coerce})$$

6.4.3 Remark: If $\Gamma \vdash_A\ e : \xi$, then all variables occurring free in ξ occur in Γ.

Both the algorithmic and the original typing rules are capable of producing derivations resulting in unsolvable types. Whilst it would be possible to restrict the constraints introduced in a rule to be solvable, such local conditions will not suffice to ensure that a final derived type is solvable, since the union of two solvable constraint sets is not necessarily solvable. Thus the intent is to derive the type, then perform a global solvability check.

There are also implications for completeness: there is more freedom to work with unsolvable constraint sets in the definitive system than in the algorithmic one – but such derivations aren't interesting in practice: it suffices that the algorithmic rules are complete with respect to the definitive rules in the case where the type derived using the latter is solvable.

Soundness of Type Inference

6.4.4 Proposition: If $\Gamma \vdash_A\ e : \xi$, then $\Gamma\ \vdash\ e : \xi$.

Proof: Notice that weakening on constraint sets is admissible in the definitional type system, so long as the TypedAbs constraint on appearances of base types is not violated. So we can proceed by induction on the derivation of $\Gamma \vdash_A\ e : \xi$: each rule in the algorithmic derivation can be translated into one or more rules from the definitional system, typically the equivalent rule for the syntactic form together with an instance of Gen.

A:Taut An instance of Taut

A:Coerce An instance of Coerce followed by an instance of Gen, the former being justified by the result from proposition(6.3.7) that if $\mathit{factor}(\Gamma, \kappa, \sigma) = C$, then $\Gamma, C\ \vdash\ \kappa \leqslant \sigma$.

A:Abs An instance of Abs together with an instance of Gen.

A:TypedAbs An instance of TypedAbs, together with an instance of Gen.

A:AppArrow If $\Gamma \vdash_A\ e : \xi_1$ and open $\xi_1 = (\overline{\alpha}, C_1, \sigma{\rightarrow}\xi_1')$, then by a sequence of Inst rules we obtain $\Gamma; C_1\ \vdash\ e : \sigma{\rightarrow}\xi_1'$; And if $\Gamma \vdash_A\ e_2 : \xi$, and $\mathit{factor}(\Gamma, \xi_2, \sigma) = C_2$, $\Gamma; C_2\ \vdash\ \xi_2 \leqslant \sigma$. An application of App and one of Gen suffices.

A:AppVar Similarly, by a use of Inst, App, and Gen

A:Let By an instance of Let and one of Gen □

Minimality of Type Inference

In order to obtain a minimal typing property, we need an ordering on complex type schemes, and we shall choose the obvious one:

6.4.5 Definition: We define the relation $\Gamma \vdash \xi \preccurlyeq \kappa$ to be true iff for any σ such that $\Gamma \vdash \kappa \leqslant \sigma$, $\Gamma \vdash \xi \leqslant \sigma$. It is easy to see that $\preccurlyeq$ is transitive, and that $\Gamma \vdash \kappa \preccurlyeq \sigma$ iff $\Gamma \vdash \kappa \leqslant \sigma$.
We extend $\preccurlyeq$ to environments:

$$\begin{array}{lcll} \bullet & \preccurlyeq & \bullet & \text{The empty environment} \\ \Gamma_1, \mathsf{a}(\leqslant \tau) & \preccurlyeq & \Gamma_2, \mathsf{a}(\leqslant \tau) & \text{if } \Gamma_1 \preccurlyeq \Gamma_2 \\ \Gamma_1, x{:}\xi & \preccurlyeq & \Gamma_2, x : \kappa & \text{if } \Gamma_1 \preccurlyeq \Gamma_2 \text{ and } \Gamma_1 \vdash \xi \preccurlyeq \kappa \end{array}$$

6.4.6 Remark: Note that the second clause in the final case is equivalent to $\Gamma_2 \vdash \xi \preccurlyeq \kappa$, since in a subtyping context only the subtype bindings are relevant, and those are the same in each environment,

6.4.7 Lemma [Weakening for typing derivations]: Suppose $\Gamma = \Gamma_1, \Gamma_2$ is such that $\Gamma' = \Gamma_1, \mathsf{a}(\leqslant\tau), \Gamma_2$ is well-formed.

- If $\Gamma; C \vdash e : \kappa$ and a does not occur in Γ, κ, or C, Then $\Gamma'; C \vdash e : \kappa$ in a derivation of the same shape.
- If $\Gamma \vdash_A e : \xi$ and a does not occur in Γ or ξ, then $\Gamma' \vdash e : \xi$ in a derivation of the same shape.

Proof: Straightforward by induction on the derivation, using proposition(6.3.1) and an auxiliary renaming lemma in the style of lemma(5.1.6). □

6.4.8 Proposition: Suppose $\Gamma; C \vdash e : \kappa$ and there is a Γ-substitution θ_s such that $\Gamma \vdash \theta_s C$. Suppose further that we have some Γ_a and Γ_a-substitution θ_a such that $(\theta_s \circ \theta_a)\Gamma_a \preccurlyeq \theta_s \Gamma$ and $\Gamma_a \vdash_A e : \xi$. Then $(\theta_s \circ \theta_a)\xi \preccurlyeq \theta_s \kappa$.

Proof: Suppose we have σ_s such that $\Gamma \vdash \theta_s \kappa \leqslant \sigma_s$. We will demonstrate that $\Gamma \vdash (\theta_s \circ \theta_a)\xi \leqslant \sigma_s$. Note that it suffices to demonstrate the proposition only for those ρ not quantified at outermost level, for we can then deduce the result for any σ_s. For suppose the result holds for all such ρ, and we have $\sigma_s =$

$\forall\beta_1(\leqslant\tau_1)\ldots\beta_n(\leqslant\tau_n).\sigma'_s$, where σ'_s is not quantified at outermost level, and we write ρ_s for $[\mathtt{b}_i/\beta_i]\sigma'_s$ and

$$\Gamma, \mathtt{B} = \Gamma, \mathtt{b}_1(\leqslant\tau_1), \mathtt{b}_2(\leqslant[\mathtt{b}_1/\beta_1]\tau_2), \ldots \mathtt{b}_n(\leqslant[\mathtt{b}_1/\beta_1, \ldots \mathtt{b}_{n-1}/\beta_{n-1}]\tau_n)$$

By lemma(6.4.7) we have a derivation of $\Gamma, \mathtt{B}; C \vdash e : \kappa$ which is of the same shape as that of $\Gamma; C \vdash e : \kappa$, and one of $\Gamma_a, \mathtt{B} \vdash e : \xi$. And since we have $\Gamma \vdash \theta_s\kappa \leqslant \sigma_s$, a canonical derivation must include $\Gamma, \mathtt{B} \vdash \theta_s\kappa \leqslant \rho_s$. Thus we can deduce $\Gamma, \mathtt{B} \vdash e : (\theta_s \circ \theta_a)\xi \leqslant \rho_s$.

But ξ contains no occurrences of atoms in $\mathtt{B}$, and since θ_s and θ_a are Γ-well formed, none are introduced by the substitution. So none of the $\mathtt{B}$ occur in $(\theta_s \circ \theta_a)\xi$, and thus we have $\Gamma \vdash (\theta_s \circ \theta_a)\xi \leqslant \sigma_s$ as required.

So suppose we have $\Gamma \vdash \theta_s\kappa \leqslant \rho_s$. We proceed by induction on the height of the derivation of $\Gamma; C \vdash e : \kappa$ by consideration of the last rule.

Taut

$$\frac{\Gamma(x) = \kappa}{\Gamma; C \vdash x : \kappa}$$

The last rule in the algorithmic derivation must be A:Taut. Since $(\theta_s \circ \theta_a)\Gamma_a \preccurlyeq \theta_s\Gamma$, if $\Gamma_a(\alpha) = \xi$, we have $\Gamma \vdash (\theta_s \circ \theta_a)(\xi) \preccurlyeq \theta_s\kappa$.

Abs The last rule in the algorithmic derivation must be A:Abs

$$\frac{\Gamma, x{:}\tau; C \vdash e : \kappa}{\Gamma; C \vdash \lambda x.e : \tau\rightarrow\kappa} \qquad \frac{\Gamma_a, x{:}\alpha \vdash_A e : \xi}{\Gamma_a \vdash_A \lambda x.e : \forall\alpha.\alpha\rightarrow\xi}$$

If we have a derivation of $\Gamma \vdash \theta_s(\tau\rightarrow\kappa) \leqslant \rho_s$, then ρ_s must be of the form $\rho_1\rightarrow\rho_2$ (the only rule applicable in a canonical derivation being Arrow) and we must have $\Gamma \vdash \theta_s\kappa \leqslant \rho_2$. Thus writing $\theta'_a = [\tau/\alpha] \circ \theta_a$, we have $(\theta_s \circ \theta'_a)(\Gamma_a, x{:}\alpha) \preccurlyeq \theta_s(\Gamma, x{:}\tau)$, and by the induction hypothesis $\Gamma \vdash (\theta_s \circ \theta'_a)\xi \leqslant \rho_2$. We also have $\Gamma \vdash \rho_1 \leqslant \theta_s\tau$, and as $\tau = \theta'_a\alpha$, we have $\Gamma \vdash \rho_1 \leqslant (\theta_s \circ \theta'_a)\alpha$. Thus we can deduce

$$\Gamma \vdash (\theta_s \circ \theta'_a)(\alpha\rightarrow\xi) \leqslant \rho_1\rightarrow\rho_2$$

and hence

$$\Gamma \vdash (\theta_s \circ \theta_a)(\forall\alpha.\alpha\rightarrow\xi) \leqslant \rho_1\rightarrow\rho_2$$

TypedAbs

$$\frac{\begin{array}{c}\Gamma, \mathtt{A}, x{:}[\mathtt{a}_i/a_i]\sigma; C \vdash [\mathtt{a}_i/a_i]e : \kappa \\ \Gamma, \mathtt{A}; C \vdash \kappa \leqslant \sigma' \text{ for some } \sigma' \\ \text{no } \mathtt{a}_i \text{ occurs in } C \qquad \alpha_i \text{ fresh}\end{array}}{\Gamma; C \vdash \lambda x{:}\sigma.e : [\alpha_i/a_i]\sigma\rightarrow[\alpha_i/\mathtt{a}_i]\kappa}$$

The last rule in the algorithmic derivation must be A:TypedAbs

$$\frac{\begin{array}{c}\Gamma_a, \mathtt{A}, x{:}[\mathtt{a}_i/a_i]\sigma \vdash_A \ [\mathtt{a}_i/a_i]e : \xi \\ \Gamma_a, \mathtt{A} \vdash_A \ \xi \text{ feasible}\end{array}}{\Gamma_a \vdash_A \ \lambda x{:}\sigma.e : \forall \alpha_i.[\alpha_i/a_i]\sigma {\rightarrow} [\alpha_i/\mathtt{a}_i]\xi}$$

where $\mathrm{fv}(\sigma) = a_1 \ldots a_n$. and $\mathtt{A}$ is $\mathtt{a}_1 \ldots \mathtt{a}_n$.

Suppose we have $\Gamma; C \ \vdash \ [\alpha_i/a_i]\sigma{\rightarrow}[\alpha_i/\mathtt{a}_i]\kappa \leqslant \rho$, then ρ must be of the form $\rho_1{\rightarrow}\rho_2$, and by induction on the proof it is easy to show that

$$\Gamma, \mathtt{A}; C \ \vdash \ [\mathtt{a}_i/a_i]\sigma{\rightarrow}\kappa \leqslant [\mathtt{a}_i/\alpha_i](\rho_1{\rightarrow}\rho_2)$$

since no α_i occurs in C. Then by the induction hypothesis, we have

$$\Gamma, \mathtt{A}; \ \vdash \ [\mathtt{a}_i/a_i]\sigma{\rightarrow}\xi \leqslant [\mathtt{a}_i/\alpha_i](\rho_1{\rightarrow}\rho_2)$$

and thus

$$\Gamma, \mathtt{A}; \ \vdash \ \forall \alpha_i.[\alpha_i/a_i]\sigma{\rightarrow}[\alpha_1/\mathtt{a}_i]\xi \leqslant [\mathtt{a}_i/\alpha_i](\rho_1{\rightarrow}\rho_2)$$

(using ∀-left′), and then by ∀-right, instantiation, and transitivity, we have:

$$\Gamma \ \vdash \ \forall \alpha_i.[\alpha_i/a_i]\sigma{\rightarrow}[\alpha_1/\mathtt{a}_i]\xi \leqslant \rho_1{\rightarrow}\rho_2$$

App

$$\frac{\Gamma; C \ \vdash \ e_1 : \sigma{\rightarrow}\kappa \qquad \Gamma; C \ \vdash \ e_2 : \sigma}{\Gamma; C \ \vdash \ e_1 e_2 : \kappa}$$

In any derivation of $\Gamma_a \vdash_A \ e_1 : \xi_1$, the final rule must be either A:AppVar or A:AppArrow depending on the syntactic form of ξ_1. Suppose the former:

$$\frac{\begin{array}{c}\Gamma_a \vdash_A \ e_1 : \xi_1 \qquad \Gamma_a \vdash_A \ e_2 : \xi_2 \qquad \text{open } \xi_1 = (\delta_1, D_1, \alpha) \\ \mathit{factor}(\Gamma_a, \xi_2, \alpha_1) = D_2 \qquad \alpha_1, \alpha_2 \text{ fresh}\end{array}}{\Gamma_a \vdash_A \ e_1 e_2 : \mathrm{close}(\Gamma_a, D_1 \cup D_2 \cup \{\alpha {\leqslant} \alpha_1 {\rightarrow} \alpha_2\}, \alpha_2)}$$

Let $\overline{\delta}_1$ be $\mathrm{fv}(D_1) - \mathrm{fv}(\Gamma_a)$ and $\overline{\delta}_2 = \mathrm{fv}(D_2) - \mathrm{fv}(\Gamma_a)$. These are disjoint since the only free variables ξ_1 and ξ_2 share are those occurring in Γ_a.

Since $\Gamma \ \vdash \ \theta_s\kappa \leqslant \rho_s$, we have $\Gamma \ \vdash \ \theta_s(\sigma{\rightarrow}\kappa) \leqslant \theta_s\sigma{\rightarrow}\rho_s$, so from the induction hypothesis

$$\Gamma \ \vdash \ (\theta_s \circ \theta_a)\xi_1 \leqslant \theta_s\sigma{\rightarrow}\rho_s$$

Now by proposition)6.4.2 there is a substitution θ, extending $(\theta_s \circ \theta_a)$ by the variables in $\overline{\delta}_1$ such that $\Gamma \ \vdash \ \theta D_1$, and $\Gamma \ \vdash \ \theta\alpha \leqslant \theta_s\sigma{\rightarrow}\rho_s$.

So $\theta\alpha$ must either be an arrow type $\tau_1{\rightarrow}\tau_2$ or a base type which promotes to an arrow type $\tau_1{\rightarrow}\tau_2$. In either case, setting $\theta' = [\tau_i/\alpha_i] \circ \theta$, we have $\Gamma \vdash \theta_s\sigma \leqslant \theta'\alpha_1$, and from the induction hypothesis we have $\Gamma \vdash \theta'\xi_2 \leqslant \theta_s\sigma$, so by transitivity, $\Gamma \vdash \theta'\xi_2 \leqslant \theta'\alpha_1$.

So by proposition (6.3.7) there is an extension θ'' of θ' by the variables in $\overline{\delta}_2$ such that $\Gamma \vdash \theta'' D_2$. Thus we have θ'' an extension of $(\theta_s \circ \theta_a)$ by the variables $\{\alpha_1, \alpha_2\} \cup \overline{\delta}_1 \cup \overline{\delta}_2$ such that $\Gamma \vdash \theta''(D_1 \cup D_2 \cup \{\alpha \leqslant \alpha_1{\rightarrow}\alpha_2\})$, and $\Gamma \vdash \theta''\alpha_2 \leqslant \rho_s$. The result follows immediately.

The A:AppArrow case is similar.

Let

$$\frac{\Gamma; C_1 \vdash e_1 : \kappa_1 \qquad \Gamma, x{:}\kappa_1; C_1 \vdash e_2 : \kappa_2 \qquad \text{strip } \kappa_1 = (C_2, \sigma)}{\Gamma; C_1 \cup C_2 \vdash \texttt{let } x = e_1 \texttt{ in } e_2 : \kappa_2}$$

The last rule in the algorithmic derivation must be A:Let.

$$\frac{\Gamma_a \vdash_A e_1 : \xi_1 \qquad \Gamma_a, x{:}\xi_1 \vdash_A e_2 : \xi_2 \qquad \text{strip } \xi_1 = (D, \sigma')}{\Gamma_a \vdash_A \texttt{let } x = e_1 \texttt{ in } e_2 : \text{close}(\Gamma_a, D, \xi_2)}$$

Since $\Gamma \vdash \theta_s C_1$, from the induction hypothesis we have $\Gamma \vdash (\theta_s \circ \theta_a)\xi_1 \preccurlyeq \theta_s\kappa_1$, and thus $\Gamma \vdash (\theta_s \circ \theta_a)(\Gamma_a, x{:}\xi_1) \preccurlyeq \theta_s(\Gamma, x{:}\kappa_1)$. Now since $\Gamma \vdash \theta_s C_2$, $\theta_s\kappa_1$ must be feasible, and thus there is some σ' not outermost quantified such that $\Gamma \vdash \theta_\kappa \leqslant \sigma'$. Using the induction hypothesis we obtain $\Gamma \vdash (\theta_s \circ \theta_a)\xi_1 \leqslant \sigma'$, so $D = \text{strip } \xi_1$ is solvable. Thus we obtain $\Gamma \vdash \text{close}(\Gamma_a, D, \xi_2) \leqslant \rho_s$ by $\forall_C$-left′.

Coerce The last rule in the algorithmic derivation must be A:Coerce

$$\frac{\begin{array}{c}\Gamma; C \vdash e_1 : \sigma \\ \sigma \in \sigma(\Gamma)\end{array}}{\Gamma; C \vdash (e_1 : \sigma) : \sigma} \qquad \frac{\begin{array}{c}\Gamma_a \vdash_A e_1{:}\xi \\ \mathit{factor}(\Gamma_a, \xi, \sigma) = C \\ \sigma \in \sigma(\Gamma_a)\end{array}}{\Gamma_a \vdash_A (e_1{:}\sigma) : \text{close}(\Gamma, C, \sigma)}$$

If $\overline{\alpha}$ is $\text{fv}(C) - \text{fv}(\Gamma)$, by proposition(6.3.7) there is an extension θ of $(\theta_s \circ \theta_a)$ by the variables in $\overline{\alpha}$ such that $\Gamma_a \vdash_A \theta C$. Thus $\Gamma \vdash (\theta_s \circ \theta_a)(\forall\alpha\backslash C.\sigma) \leqslant \theta_s\sigma$. The result follows by transitivity.

Sub

$$\frac{\Gamma; C \vdash e : \kappa \qquad \Gamma; C \vdash \kappa \leqslant \sigma}{\Gamma; C \vdash e : \sigma}$$

Suppose $\Gamma_a \vdash_A e : \xi$. Since $\Gamma \vdash \theta_s C$, we have $\Gamma \vdash \theta_s\kappa \leqslant \theta_s\sigma$, and by assumption we have $\theta_s\sigma \leqslant \rho_s$, so by transitivity we have $\Gamma \vdash \theta_s\kappa \leqslant \rho_s$. The result follows from the induction hypothesis.

Gen

$$\frac{\Gamma; C_1 \cup C_2 \;\vdash\; e : \kappa \qquad \overline{\alpha} \cap (\mathrm{fv}(\Gamma) \cup \mathrm{fv}(C_1)) = \varnothing}{\Gamma; C_1 \;\vdash\; e : \forall\overline{\alpha}\backslash C_2.\kappa}$$

Suppose $\Gamma_a \vdash_A e : \xi$. Since we have $\Gamma \;\vdash\; \theta_s(\forall\overline{\alpha}\backslash C_2.\kappa) \leqslant \rho_s$, the last rule in the canonical derivation must be $\forall_{C\leqslant}$-LEFT′, so there must be an extension θ'_s of θ_s by the variables in $\overline{\alpha}$ such that $\Gamma \;\vdash\; \theta'_s C_2$ and $\Gamma \;\vdash\; \theta'_s \kappa \leqslant \rho_s$. Since $\Gamma \;\vdash\; \theta_s C_1$ also, the result follows from the induction hypothesis.

Inst

$$\frac{\Gamma; C_1 \;\vdash\; e : \forall\overline{\alpha}\backslash C_2.\kappa}{\Gamma; C_1 \cup C_2 \;\vdash\; e : \kappa} \qquad \text{(Inst)}$$

Suppose we have $\Gamma \vdash_A e : \xi$ and $\Gamma \;\vdash\; \theta_s \kappa \leqslant \rho_s$. Then by $\forall_C$-LEFT′ we have $\Gamma \;\vdash\; \theta_s(\forall\overline{\alpha}\backslash C_2.\kappa) \leqslant \rho_s)$. The result follows from the induction hypothesis. □

6.4.9 Corollary [Minimality of Type Inference]: If $\Gamma \;\vdash\; e : \kappa$, and $\Gamma \vdash_A e : \xi$, then $\Gamma \;\vdash\; \xi \preccurlyeq \kappa$.

Proof: Take $\Gamma_a = \Gamma$, θ_a and θ_s to be identity substitutions in proposition(6.4.8). □

Completeness of Type Inference

6.4.10 Proposition: Suppose $\Gamma; C \;\vdash\; e : \kappa$ is feasible, so we have $\Gamma \;\vdash\; \theta_s C$ and $\Gamma \;\vdash\; \theta_s \kappa \leqslant \rho$ where ρ is not outermost quantified. Then for any feasible Γ_a and Γ_a-substitution θ_a such that $(\theta_s \circ \theta_a)\Gamma_a \preccurlyeq \theta_s \Gamma$, there is some ξ such that $\Gamma_a \vdash_A e : \xi$.

Proof: We will proceed by induction on the height of the derivation of $\Gamma \;\vdash\; e : \kappa$ by cases on the last rule.

Taut

$$\frac{\Gamma(x) = \kappa}{\Gamma; C \;\vdash\; x : \kappa}$$

The algorithmic derivation is simply an instance of A:Taut.

Abs

$$\frac{\Gamma, x{:}\tau; C \;\vdash\; e : \kappa}{\Gamma; C \;\vdash\; \lambda x.e : \tau{\to}\kappa}$$

Let α be fresh. We define $\theta'_a = [\tau/\alpha] \circ \theta_a$, so that $(\theta_s \circ \theta'_a)(\Gamma_a, x{:}\alpha) \preccurlyeq \theta_s(\Gamma, x{:}\tau)$. By the induction hypothesis, we have a derivation of $\Gamma_a, x{:}\alpha \;\vdash\; e : \xi$. So we may apply A:Abs to obtain $\Gamma_a \vdash_A \lambda x.e : \forall\alpha.\alpha{\to}\xi$.

TypedAbs

$$\frac{\begin{array}{c}\Gamma, \mathtt{A}, x{:}[\mathtt{a}_i/a_i]\sigma; C \ \vdash\ [\mathtt{a}_i/a_i]e : \kappa \\ \Gamma, \mathtt{A}; C \ \vdash\ \kappa \leqslant \sigma' \ \text{ for some } \sigma' \\ \text{no } \mathtt{a}_i \text{ occurs in } C \qquad \alpha_i \text{ fresh}\end{array}}{\Gamma; C \ \vdash\ \lambda x{:}\sigma.e : [\alpha_i/a_i]\sigma{\to}[\alpha_i/\mathtt{a}_i]\kappa}$$

where $\mathrm{fv}(\sigma) = a_1 \ldots a_n$. and $\mathtt{A}$ is $\mathtt{a}_1 \ldots \mathtt{a}_n$.

Let θ be a $(\Gamma, \mathtt{A})$-substitution such that $\Gamma, \mathtt{A} \ \vdash\ \theta C$. Then $\Gamma, \mathtt{A} \ \vdash\ \theta C$ and $\Gamma, \mathtt{A} \ \vdash\ \theta\kappa \leqslant \theta\sigma'$.

Thus by using proposition(6.4.8), we must have $\Gamma, \mathtt{A}, x{:}[\mathtt{a}_i/a_i]\sigma \ \vdash\ [\mathtt{a}_1/a_i]e : \xi$ such that $\Gamma, \mathtt{A} \ \vdash\ (\theta_s \circ \theta)\xi \leqslant \theta\sigma_1$. Now since Γ and Γ_a are the same with respect to the subtyping assertions they prove, we must have $\Gamma_a, \mathtt{A} \vdash \xi$ feasible. So we may apply A:TypedAbs.

App

$$\frac{\Gamma; C \ \vdash\ e_1 : \sigma{\to}\kappa \qquad \Gamma; C \ \vdash\ e_2 : \sigma}{\Gamma; C \ \vdash\ e_1 e_2 : \kappa}$$

Depending on the syntactic form of open ξ_1, the algorithmic rule will either be A:AppVar or A:AppArrow, so suppose open $\xi_1 = (D_1, \alpha)$ and it is the latter case. By the induction hypothesis, we have a derivation of $\Gamma_a \vdash_A e_1 : \xi_1$ and from proposition(6.4.8) we have $\Gamma \ \vdash\ \theta_a\xi_1 \preccurlyeq \sigma{\to}\kappa$. Since the canonical derivation of $\Gamma \ \vdash\ \theta_a\xi_1 \preccurlyeq \sigma{\to}\kappa$ must finish with an Arrow rule followed by some sequence of $\forall$-left$'$ rules, there must be an extension θ'_a of θ_a such that $\Gamma \ \vdash\ \sigma \leqslant \theta'_a\alpha_1$. Now since $\Gamma \ \vdash\ \theta_a\xi_2 \leqslant \sigma_1$, we have $\Gamma \ \vdash\ \theta'_a\xi_2 \leqslant \theta'_a\alpha_1$ by transitivity, and thus *factor*$(\Gamma_a, \xi_2, \alpha_1)$ succeeds and we may apply A:AppVar.

The AppArrow case is similar.

Let

$$\frac{\Gamma; C_1 \ \vdash\ e_1 : \kappa_1 \qquad \Gamma, x{:}\kappa_1; C_1 \ \vdash\ e_2 : \kappa_2 \qquad \text{strip } \kappa_2 = (C_2, \sigma)}{\Gamma; C_1 \cup C_2 \ \vdash\ \mathtt{let}\ x = e_1\ \mathtt{in}\ e_2 : \kappa_2}$$

From the induction hypothesis, we have $\Gamma_a \vdash_A e_1 : \xi_1$ such that $\Gamma \ \vdash\ (\theta_s \circ \theta_a)\xi_1 \preccurlyeq \theta_s\kappa_1$. Thus we have $(\theta_s \circ \theta_a)(\Gamma_a, x{:}\xi_1) \preccurlyeq \theta_s(\Gamma, x{:}\kappa_1)$. And using the induction hypothesis again we have $\Gamma_a, x{:}\xi_1 \vdash_A e : \xi_2$, and we may apply A:Let.

Coerce

$$\frac{\Gamma; C \ \vdash\ e_1 : \sigma \qquad \sigma \in \sigma(\Gamma)}{\Gamma; C \ \vdash\ (e_1 : :\sigma) : \sigma}$$

By the induction hypothesis we have $\Gamma_a \vdash_A \ \xi$ with $\Gamma \ \vdash \ \theta_a\xi \leqslant \sigma$. Thus since $\sigma \in \sigma(\Gamma_a)$, $\mathit{factor}(\Gamma_a, \xi, \sigma)$ succeeds and we may apply A:Coerce.

Sub

$$\frac{\Gamma; C \ \vdash \ e : \kappa \qquad \Gamma; C \ \vdash \ \kappa \leqslant \sigma}{\Gamma; C \ \vdash \ e : \sigma}$$

Since $\Gamma \ \vdash \ \theta_s C$, we have $\Gamma \ \vdash \ \theta_s\kappa \leqslant \theta_s\sigma$ and $\Gamma \ \vdash \ \theta_s\sigma \leqslant \rho_s$. Thus by transitivity we have $\Gamma \ \vdash \ \theta_s\kappa \leqslant \rho_s$, and the result follows from the induction hypothesis.

Gen

$$\frac{\Gamma; C_1 \cup C_2 \ \vdash \ e : \kappa \qquad \overline{\alpha} \cap (\mathrm{fv}(\Gamma) \cup \mathrm{fv}(C_1)) = \varnothing}{\Gamma; C_1 \ \vdash \ e : \forall\overline{\alpha}\backslash C_2.\kappa}$$

By consideration of the canonical derivation of $\Gamma \ \vdash \ \forall\overline{\alpha}\backslash C_2.\kappa$, we have an extension θ'_s of θ_s by $\overline{\alpha}$ such that $\Gamma \ \vdash \ \theta'_s(C_1 \cup C_2)$. The result follows from the induction hypothesis.

Inst

$$\frac{\Gamma; C_1 \ \vdash \ e : \forall\overline{\alpha}\backslash C_2.\kappa}{\Gamma; C_1 \cup C_2 \ \vdash \ e : \kappa}$$

We have $\Gamma \ \vdash \ \theta_s\kappa \leqslant \rho$, so by $\forall_C$-left′, we have $\Gamma \ \vdash \ \theta_s(\forall\overline{\alpha}\backslash C_2.\kappa) \leqslant \rho$. The result follows from the induction hypothesis. □

6.4.11 Corollary [Completeness of Type Inference]: If $\Gamma; C \ \vdash \ e : \kappa$ is feasible. Then $\Gamma \vdash_A \ e : \xi$ for some Γ_a-feasible ξ.

Proof: Taking $\Gamma_a = \Gamma$ and θ_a the identity, the result is immediate, apart from the Γ_a-feasibility of ξ. But if $\Gamma \ \vdash \ e : \kappa$ is feasible, then there is some θ and σ such that $\Gamma \ \vdash \ \theta C$ and $\Gamma \ \vdash \ \theta\kappa \leqslant \sigma$. By proposition(6.4.8) $\Gamma \ \vdash \ \xi \preccurlyeq \kappa$, and thus $\Gamma \ \vdash \ \theta\xi \leqslant \sigma$, so ξ is Γ-feasible. □

6.5 Examples

The type inference algorithm as given, like the inference algorithm for $\mathrm{ML}_\leqslant$ given in the introduction, is "raw", in the sense that it produces unsimplified typings. And similarly to $\mathrm{ML}_\leqslant$, this is the principal barrier to presenting comprehensible examples. Consider:

```
let x = fun (f: All(A) All(B<A) B -> B -> B)
   fun(a) fun (b) fun (c) fun (d)
     (f a c, f b d);
```

produces (reformatted for legibility) the expected typing:

```
x : All(A)(All(B<A) B->B->B)->
     All('j) 'j->
       All('k) 'k->
         All('l) 'l->
           All('m) 'm->
             All('w,'v,'u,'x;
                 'u < 'w, 'u < A, 'j < 'u, 'l < 'u,
                 'x < 'v, 'x < A, 'k < 'x, 'm < 'x)  Pair 'w 'v
```

and the application

```
let x = pairApp max 10 -7  20 -3;
```

typechecks successfully. Obviously, like $\mathrm{ML}_{\leqslant}$, the system is unusable without constraint simplification. In cases such as the above, where a sequence of abstractions is followed by an abstraction-free function body, it can be observed that constraints accumulate at the outermost type inside the abstractions; in such cases we may reasonably expect scoping to have little impact on simplification; however, when the value returned by the function is another function computed at some depth inside the function body, for example

```
let x = fun (h)
  let z = fun(g) g h
  in z
  end
;
```

assigned the principal type is

```
x : All('b) 'b->
      All('g,'i,'h; 'g < 'i->'h, 'b < 'i)
         All('d) 'd->
            All('e,'f; 'd < 'e->'f, 'b < 'e) 'f
```

It is possible to combine the two adjacent universal quantifiers, but this does not rid us of the constrained quantifier nesting.

6.6 Conclusion

It is clear from even the small examples that simplification is required to make $\mathrm{ML}_{\forall\leqslant}$ a practical type system. While we have not investigated the issue in detail, it appears that scoping significantly complicates the more powerful simplifications available. There is little difficulty with identifying two types if simplification indicates equality, but the opportunity arises less often than in $\mathrm{ML}_{\leqslant}$, since for example it is only complete to identify a variable α with its least upper bound τ if every possible value for τ is a possible value for α, in other words, if for every free variable β of τ, every base type which can occur in a well-scoped substitution for β can also occur in a well-scoped substitution for α. Developing a theory of simplification would require a instance relation to formalise equivalence on judgements, but a simple definition remains elusive. Another way to make the approach here scalable is to annotate top-level function definitions with their intended type, against which their inferred type can be checked.

Laufer and Odersky prove their type system equivalent to system F, in the sense that any type derivation in system F can be encoded in their system extended by type operator definitions of the form

$$\texttt{newtype T}v_1 \ldots v_n = \sigma$$

with corresponding injection and projection terms. The encoding (we omit the details, which are not particularly enlightening) uses a type operator to encapsulate each instance of universal quantification, and the operators must be defined with respect to the $\mathrm{F}_{\leqslant}$ derivation rather than just the term. Since we are restricted to structural subtyping and distinguish between bounded and unbounded quantification, it is difficult to see how we might embed $\mathrm{F}_{\leqslant}$; however an embedding is possible for an $\mathrm{F}_{\leqslant}$ variant which uses the subtyping rules

$$\frac{\Delta, X \vdash_F \; S_1 \leqslant S_2}{\Delta \vdash_F \; \forall(X)S_1 \leqslant \forall(X)S_2} \qquad (\forall)$$

$$\frac{\Delta \vdash_F \; S_2 \leqslant S_1 \qquad \Delta, X \; \vdash \; S_1' \leqslant S_2'}{\Delta \vdash_F \; \forall(X{\leqslant}S_1)S_1' \leqslant \forall(X{\leqslant}S_2)S_2'} \qquad (\forall_{\leqslant})$$

with a corresponding separation at the term level between bounded and unbounded type abstraction.

This version of $\mathrm{F}_{\leqslant}$ is similar to the system $\mathrm{F}_{\leqslant}^{\top}$ ([8, 9]) whose flaw, from which $\mathrm{ML}_{\forall\leqslant}$ does not suffer, is that it lacks a minimal type property. Thus while it is not possible to uniformly encode $\mathrm{F}_{\leqslant}$ programs as $\mathrm{ML}_{\forall\leqslant}$ programs, we might perhaps justifiably claim that much of what can be done in raw $\mathrm{F}_{\leqslant}^{\top}$ can be done in $\mathrm{ML}_{\forall\leqslant}$.

Conclusion

Summary of Results

Subtyping provides a convenient framework in which to express relations of subsumption and implicit coercion. Here we summarise our results and consider their relevance to subtyping systems in general.

$\top$-less Models

Much of this thesis has been devoted to the study of the properties of subtyping over Helly posets. Up to now, approaches using complex type systems (such as those of Smith [16] or Pottier [40]) have been based on the "one big lattice" approach, extending the ordering given by the subtyping constructors with $\top$ and $\bot$. The merit of these types in themselves is an open issue. $\top$ constitutes a convenient implementation mechanism for typing features such heterogeneous lists, and so allows restricted sacrifice of type safety, but in a strong typing context it is hard to see the intrinsic value of knowing that a term inhabits, requires an argument, or produces a result which is in the type containing all elements, much less the type containing no elements. Perhaps something of the utility of $\top$ and $\bot$ in the systems of Smith and Pottier can be inferred from that fact that their approaches incorporate these elements in the models of their subtype systems, but not the syntax.

Our results allow us to construct a good case for Helly posets as a feasible point in the design space for subtype orderings. We have shown that as well as encompassing many classes of natural orderings on atomic types, Helly posets are closed under type formation using many constructors of interest, so they are versatile enough to capture many interesting orderings and some of the ways in which we might wish to extend them. We have demonstrated how solvability may be determined for both finite and regular types with both structural and non-structural constructors. And we have shown in a simple context that Helly posets enable the use of semantic information to provide a powerful simplification system.

The choice of ordering in a subtyping system is made on grounds of tractability, expressive power, and strength. Operations on Helly posets are typically analogous to operations on lattices: distance closure over the former corresponds with constraint closure over the latter. So it seems reasonable to expect low-diameter Helly posets to be approximately as efficient as lattices in most cases. In terms of expressive power, there seems very little to choose between the two. And in terms of preventing type errors, they lack the weakness that comes from including $\top$ and $\bot$, and so better capture the intuitions behind complex type systems.

Bounded Quantification

We have also studied the issue of bounded quantification, a natural outgrowth of the extensibility properties of Helly posets. As well as the natural application in signature-checking and top-level coercion to keep constraint sets manageable, bounded type schemes have a principality property that can be seen as analogous to that of mixed-prefix unification, which we exploited to implement a type system with annotations containing subtypes. Whilst the question of how principal constraint sets can be generated for subtyping constrained type schemes is a difficult one, our results in this area constitute an answer to the question of how subtyping and annotation can be made to interact. However, the practical utility of such a complex type system is an open issue, particularly until the question of simplification is resolved.

Directions for Future Work

There are several issues which we have chosen not to cover in this thesis, which would constitute natural and interesting extensions of the work.

Simplification

After solvability, simplification is the most fundamental operation necessary to successful use of subtyping. Although we have given a method for simplification in the atomic case, the approach needs to be extended to deal with non-structural constructors in the style of [40].

Complexity

We have paid little attention to complexity issues, but the analogy between distance closure over a Helly poset and constraint closure over a lattice would seem to indicate that similar complexity arguments (such as [21]) should apply, in the same way as Benke's result [5] generalises that of Tiuryn [50] for solvability in the finite structural case.

Bibliography

[1] M. Abadi and L. Cardelli. On subtyping and matching. In Walter Olthoff, editor, *Proceedings of the 9th European Conference on Object-Oriented Programming (ECOOP'95)*, volume 952 of *LNCS*, pages 145–167, Berlin, GER, August 1995. Springer.

[2] M. Abadi and L. Cardelli. *A Theory of Objects*. Springer-Verlag, New York, 1996.

[3] A. Aiken, E. Wimmers, and J.Palsberg. Optimal representation of polymorphic types with subtyping. Technical Report UCB/CSD-96-909, University of California, Berkeley, July 1996.

[4] R. Amadio and L. Cardelli. Subtyping recursive types. In *Proceedings of the 18th ACM Symposium on Principles of Programming Languages*, pages 104–118, Orlando, Florida, january 1991. ACM Press.

[5] M. Benke. Efficient type reconstruction in the presence of inheritance. In *Mathematical Foundations of Computer Science*, pages 272–80. Springer-Verlag, 1993.

[6] M. Brandt and F. Henglein. Coinductive axiomatization of recursive type equality and subtyping. In Roger Hindley, editor, *Proc. 3d Int'l Conf. on Typed Lambda Calculi and Applications (TLCA), Nancy, France, April 2-4, 1997*, Lecture Notes in Computer Science (LNCS). Springer-Verlag, January 1997. To appear.

[7] L. Cardelli and P. Wegner. On understanding types, data abstraction, and polymorphism. *ACM Computing Surveys*, 17(4):471–522, December 1985.

[8] G. Castagna and B. Pierce. Decidable bounded quantification. In *Proceedings of the Twenty-First ACM Symposium on Principles of Programming Languages (POPL), Portland, Oregon*. ACM Press, January 1994.

[9] G. Castagna and B. Pierce. Corrigendum: Decidable bounded quantification. In *Proceedings of the Twenty-Second ACM Symposium on Principles of Programming Languages (POPL), Portland, Oregon.* ACM Press, January 1995.

[10] M. Coppo and M. Dezani-Ciancaglini. An extension of the basic functionality theory for the λ-calculus. *Notre-Dame Journal of Formal Logic*, 21(4):685–693, October 1980.

[11] P.-L. Curien and G. Ghelli. Coherence of subsumption. In *CAAP: Colloquium on Trees in Algebra and Programming.* LNCS, Springer-Verlag, 1990.

[12] P. Curtis. Constrained quantification in polymorphic type analysis. Technical Report CSL-90-1, Xerox Palo Alto Research Center, February 1990.

[13] L. Damas and R. Milner. Principal type schemes for functional programs. In *Proceedings of the 9th ACM Symposium on Principles of Programming Languages*, pages 207–212, 1982.

[14] D. Duggan. Polymorphic methods with self types for ML-like languages. Technical Report CS-95-03, University of Waterloo, 1995.

[15] J. Eifrig, S. Smith, and V. Trifonov. Sound polymorphic type inference for objects. In *OOPSLA*, 1995.

[16] J. Eifrig, S. Smith, and V. Trifonov. Type inference for recursively constrained types and its applications to OOP. In *Conference on Mathematical Foundations of Programming Languages*, 1995.

[17] A. Frey. Satisfying systems of subtype inequalities in polynomial space. Proc. 4th International Static Analysis Symposium 1997.

[18] Y. Fuh and P. Mishra. Polymorphic subtype inference: Closing the theory-practice gap. In *Proceedings of the 22nd International Conference on Theory and Practice of Software Development*, pages 167–183, Barcelona, Spain, March 1989. Springer-Verlag.

[19] F. Henglein. Syntactic properties of polymorphic subtyping. TOPPS Technical Report (D-report series) D-293, DIKU, University of Copenhagen, May 1996.

[20] F. Henglein. Breaking through the n^3 barrier: Faster object type inference. In Benjamin Pierce, editor, *Proc. 4th Int'l Workshop on Foundations*

of Object-Oriented Languages (FOOL), Paris, France, January 1997. Published electronically at URL http://www.cs.williams.edu/˜kim/FOOL/.

[21] F. Henglein and J. Rehof. The complexity of subtype entailment for simple types. TOPPS D-report D-319, DIKU, University of Copenhagen, Universitetsparken 1, DK-2100 Copenhagen, Denmark, January 1997.

[22] M. Hoang and J. Mitchell. Lower bounds on type inference with subtypes. In *Conference Record of the 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'95)*, pages 176–185, San Francisco, California, January 22–25, 1995. ACM Press.

[23] T. Jim. What are principal typings and what are they good for? In *Conference Record of POPL '96: The* 23$^{\text{rd}}$ *ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 42–53, St. Petersburg Beach, Florida, 21–24 January 1996.

[24] D. Kozen, J. Palsberg, and M. Schwartzbach. Efficient inference of partial types. *Journal of Computer and System Sciences*, 49(2):306–324, October 1994.

[25] D. Kozen, J. Palsberg, and M. Schwarzbach. Efficient recursive subtyping. In *Proc 20th Annual ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages*, pages 419–428. ACM Press, January 1993.

[26] D. Miller. Unification under a mixed prefix. *Journal of Symbolic Computation*, 14:321–358, 1992.

[27] R. Milner, M. Tofte, and R. Harper. *The Definition of Standard ML*. MIT Press, Cambridge, MA, 1991.

[28] J. Mitchell. Coercion and type inference (summary). In *Proceedings of the Eleventh ACM Symposium on Principles of Programming Languages*, pages 275–185. ACM Press, 1984.

[29] J. Mitchell. Polymorphic type inference and containment. *Information and Computation*, 76(2/3):211–249, February/March 1988. An extended version of the paper that appeared in the 1984 Semantics of Data Types Symposium, LNCS 173, pages 257–278.

[30] P. Nevermann and I. Rival. Holes in ordered sets. *Graphs and Combinatorics*, pages 339–350, 1985.

[31] M. Odersky and K. Läufer. Putting type annotations to work. In *Conference Record of POPL '96: The 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, St. Petersberg Beach, Florida*, pages 54–67, New York, N.Y., January 1996. ACM.

[32] J. Palsberg. Efficient type inference for object types. In *9th Annual IEEE Symposium on Logic in Computer Science*, pages 186–195, Paris, France, July 1994. IEEE Computer Society Press.

[33] J. Palsberg and T. Jim. Type inference in systems of recursive types with subtyping. Manuscript, 1997.

[34] J. Palsberg and S. Smith. Constrained types and their expressiveness. *ACM Transactions on Programming Languages and Systems*, 18(5):519–527, September 1996.

[35] J. Palsberg, M. Wand, and P. O'Keefe. Type inference with non-structural subtyping. *Formal Aspects of Computing*, 9:49–67, 1997.

[36] J. Peterson, K. Hammond, et al. Report on the programming language Haskell, a non-strict purely-functional programming language, version 1.3. Technical report, Yale University, May 1996.

[37] B. Pierce and D. Turner. Simple type-theoretic foundations for object-oriented programming. *Journal of Functional Programming*, 4(2):207–247, April 1994. A preliminary version appeared in Principles of Programming Languages, 1993, and as University of Edinburgh technical report ECS-LFCS-92-225, under the title "Object-Oriented Programming Without Recursive Types".

[38] Benjamin C. Pierce. *Programming with Intersection Types and Bounded Polymorphism.* PhD thesis, Carnegie Mellon University, December 1991. Available as School of Computer Science technical report CMU-CS-91-205.

[39] F. Pottier. Type inference and simplification for recursively constrained types. In *Actes du GDR Programmation 1995 (journée du pôle Programmation Fonctionnelle)*, November 1995.

[40] F. Pottier. Simplifying subtype constraints. In *Proceedings of the International Conference on Functional Programming*, pages 122–133, Paris, France, May 1996. ACM Press.

[41] V. Pratt and J. Tiuryn. Satisfiability of inequalities in a poset. *Fundamenta Infomaticae*, 28(1,2):165–182, 1996.

[42] A. Quilliot. An application of the helly property to the partially ordered sets. *J. Combinatorial Theory (A)*, 35:185–198, 1983.

[43] J. Rehof. Minimal typings in atomic subtyping. In *Proceedings POPL '97, 24th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Paris, France.* ACM Press, January 1997.

[44] D. Rémy. A case study of typechecking with constrained types: Typing record concatenation. Presented at the workshop on Advances in types for computer science at the Newton Institute, Cambridge, UK, August 1995.

[45] John Reynolds. Using category theory to design implicit conversions and generic operators. In N. D. Jones, editor, *Proceedings of the Aarhus Workshop on Semantics-Directed Compiler Generation*, number 94 in Lecture Notes in Computer Science. Springer-Verlag, January 1980.

[46] G. Smith. Polymorphic type inference with overloading and subtyping. In *TAPSOFT93*, pages 671–685, 1993.

[47] M. Sulzmann, M. Odersky, and M. Wehr. Type inference with constrained types. In Benjamin Pierce, editor, *Proc. 4th Int'l Workshop on Foundations of Object-Oriented Languages (FOOL), Paris, France*, January 1997. Published electronically at URL http://www.cs.williams.edu/˜kim/FOOL/.

[48] S. Thatte. Type inference with partial types. In Timo Lepistö and Arto Salomaa, editors, *Proceedings of the 15th International Colloquium on Automata, Languages and Programming*, volume 317 of *LNCS*, pages 615–629, Berlin, July 1988. Springer.

[49] J. Tiuryn. Subtyping over a lattice. Gödel colloquium talk, Vienna 1997.

[50] J. Tiuryn. Subtype inequalities. In *Proceedings of the 7th Annual Symposium on Logic in Computer Science*, pages 308–315, Santa Cruz, California, June 1992. IEEE Computer Society Press.

[51] J. Tiuryn and P. Urzyczyn. The subtyping problem for second-order types is undecidable. In *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science*, pages 74–85, New Brunswick, New Jersey, 27–30 July 1996. IEEE Computer Society Press.

[52] V. Trifonov and S. Smith. Subtyping constrained types. In *Proceedings of the Static Analysis Symposium*, pages 349–365, Aachen, Germany, 1996. Springer-Verlag.

[53] J.B. Wells. A direct algorithm for type inference in the rank-2 fragment of the second-order λ-calculus. In *ACM symposium on Lisp and Functional Programming*, pages 196–207, 1994.