Complexity of type reconstruction in programming languages with subtyping

Marcin Benke

Rozprawa Doktorska

Promotor: prof. dr hab. Jerzy Tiuryn

Uniwersytet Warszawski
Wydział Matematyki Informatyki i Mechaniki

Warszawa, 1997

Contents

Chapter 1

Introduction

1.1 Types and programs

In 1935, Alan Turing [Tur35] introduced a formalism for describing computable functions, now called Turing machines, from which imperative programming arose. At the same time, Alonzo Church was working on the design of lambda-calculus with similar intention of describing computable functions [Chu36]. But while Turing's approach concentrated on the notion of computation, the lambda-calculus stressed functions as the primary notion. From this calculus stems the branch of functional languages.

A language can be called functional if functions are *first-class values*, i.e. there are expressions in the language to denote them, variables can be bound to functions in the same way as they are bound to basic values, and functions can be taken as arguments or obtained as results of other functions without restrictions. Briefly, functional languages are those that contain lambda-calculus as a sublanguage.

The ancestor of all functional languages seems to be ISWIM (an acronym for If you See What I Mean) proposed some thirty years ago by Peter Landin [Lan66]. ISWIM, which consisted basically of lambda-calculus augmented with conditionals and recursive definitions, constitutes the core of contemporary functional languages such as Standard ML [Mil84, MTH89], Haskell [Hud92] or Clean. The main additions since ISWIM have been polymorphic type systems (ML) as well as inductive data types (Hope and Standard ML).

Types are an important tool in programming languages and logic which serves to classify terms according to basic properties such as denoting a number or a function. For example the integer 2 can be represented by the term "2" of type *integer*. The addition function has a type expressing that it takes two integer arguments and returns an integer as result, which we write as follows

$$+ : (\mathit{integer} \times \mathit{integer}) \rightarrow \mathit{integer}$$

Or, often using so called *currying*

$$+ : \mathit{integer} \rightarrow \mathit{integer} \rightarrow \mathit{integer}$$

One immediate advantage of types is that nonsensical expressions can be immediately considered illegal. For example, the expression

$$2 + \texttt{"abc"}$$

will not be accepted as correct because `"abc"` is not an integer, but a string of characters. Although this example may seem coarse, this kind of errors is very frequent in the development of programs (of course in most cases with more complicated expressions and types, but the essence remains the same).

Types can also be used to express information useful for efficient compilation, for example *strictness* and *neededness* in static analysis. Expressing such properties by type systems helps to abstract from implementation details of a particular analysis algorithm. Moreover, the type-based approach often leads to much more efficient algorithms than other approaches [Ben92, Jen92].

In a bit different direction, there are systems in which a type can not only prevent the formation of nonsensical expressions, but also state properties of terms. For example in the case of a term corresponding to a sorting algorithm for lists, the type can express the fact that its result is an ordered list.

On the other hand probably almost everyone learning a strongly-typed programming language (functional or imperative), experienced with a mixture of surprise and anger that one cannot say

$$\mathit{sqrt}(2)$$

in Pascal (since *sqrt* is of type $\mathit{real} \rightarrow \mathit{real}$ and the literal 2 is of type *integer*) or

$$2 + 2.0$$

in Standard ML for a similar reason. Putting implementation concerns aside, the initial reaction of surprise and anger is natural: mathematically, every integer is a real number and we should be able to use an integer wherever we are allowed to use a real. In the setting of types, this idea can be formalised by the mechanism called *subtyping*, which we discuss below.

1.2 Subtyping

One should not allow oneself to be deceived by the simplicity of the above examples; there is more to subtyping than "$2 + 2.0$", if I may coin a phrase. Subtyping is a relation that uniformly captures concepts from diverse areas of computer science. If σ and τ are sets, then $\sigma \leq \tau$ (σ *is a subtype of* τ) means that every element of σ is also an element of τ. If σ and τ are strictness properties, then every term having the property σ also has the property τ. If σ and τ are specifications, then elements satisfying σ also satisfy τ. In object-oriented programming, if σ and τ are object descriptions, then $\sigma \leq \tau$ states that where an object of interface τ is expected, it is safe to use an object with interface σ. If σ and τ are theorems, then a proof of σ is also a proof of τ.

The idea of subtypes appears quite naturally in programming languages. Informally, we can say that σ is a subtype of τ if any element of σ *can be seen as* an element of τ. We say *may be seen as* and not *is*, because this process can involve some transformation (e.g. in Pascal an integer value has to be converted to floating point representation before it can be used as real). Such transformation is called *coercion*. On the other hand, a record can be seen (in some contexts) as a record which is its initial fragment without any transformation; in a similar manner an object can be used as an object of its superclass; also considering an element of a subrange type like [1..100] as an integer requires no coercion.

1.3 Type inference, typability and type checking

Typed versions of the lambda calculus were introduced in [Cur34] and in [Chu40]. These two papers give rise to two different families of systems. In the Curry version terms are the same as in the type-free theory. Each term has a set of possible types (which may be empty, in which case we say that the term is not typable). In the Church systems the terms are type-annotated.

The Curry and Church approaches to typed lambda calculus correspond to two paradigms in programming. In the first a program may be written without typing at all. Then a compiler should check whether a type can be assigned to the program. This process is called *type inference* or *type reconstruction* if the compiler actually constructs the type. Otherwise (i.e. if the compiler only checks that some type for the program exists without actually constructing any) we talk about typability checking. An example of a language allowing this style of *implicit typing* is ML [Mil84, MTH89].

The other paradigm in programming is called *explicit typing* and corresponds to Church version of typed lambda calculi. Here every entity in a program must be declared with a type before it is used. For these languages the compiler only has to check whether all uses of declared entities are compatible with their declared types. This process is called *type checking* and is often easier than type inference or typability checking. Examples of languages in this family are Algol 68 and Pascal.

1.4 Background

The formal study of subtyping in programming languages was initiated by Reynolds [Rey80] and Cardelli [Car84], who used a lambda calculus with subtyping to model the refinement of interfaces in object-oriented languages, and also by Mitchell who first analysed subtyping in a database context [Mit83b] then abstracted out the problem of type inference in a simply-typed lambda calculus with subtyping [Mit84].

In 1992, Tiuryn coined the term “subtype inequalities” for the underlying algebraic problem and showed that the nature of ordering between base types has a critical impact on the computational complexity of the problem: it was shown that deciding satisfiability of subtype inequalities[1] (called SSI) is PSPACE-hard in general but can be decided in polynomial time for lattices.

The results of Hoang and Mitchell [HM95] as well as Frey [Fre97] have augmented the work of Tiuryn, showing that type inference in simply typed lambda calculus with subtyping is PSPACE-complete in general case. But the knowledge about tractable cases is still far from complete. This work is an attempt to narrow this gap.

Faced with the fact that the complexity of our problem depends so much on the properties of underlying ordering, it is natural to turn to the theory of partial orders and see what we can learn from it. While doing so, even a quick perusal through the literature of the subject can hardly fail to leave the reader with two important impressions:

- notions of posets “hard” and “easy” from subtyping point of view coincide with respective order-theoretic ones;
- despite decades of research and efforts of numerous scientists, the classification of even the finite partial orders is far from being complete; on the contrary, many problems have been open for years.

The second, sad constatation notwithstanding, it seems that research of subtyping can substantially benefit from the partial order theory, as we show in the following two chapters. However, as it is difficult to describe the state of research without plunging into technical terms, we defer this discussion until section 2.4.

[1]This problem is formally defined in section 2.2.

1.5 Overview of the work and results

This thesis is divided into two parts. The first one is devoted to subtype inequalities. Chapter 2 introduces basic definitions and results connected with subtype inequalities and partial orders in general. Chapter 3 is a study in application of order-theoretic methods to show that for a particular class of posets with known "generators"[2], SSI can be decided in polynomial time. The class we analyse includes the class of trees, which seems to be of importance in the study of subtyping, as trees correspond to the coercion structure of single-inheritance object systems.

When dealing with a class for which no set of generators is known, order-theoretic notions are of less use. In Chapter 4 we show how logical methods can be applied to show similar result for such a class.

The second part discusses problems concerning combining subtyping with Hindley-Milner (aka ML) type discipline. This discipline, using shallow polymorphic type schemes[3], constitutes the kernel of type systems of almost all current functional languages, including Standard ML, CAML, Haskell, Clean and Miranda. Exceptions are of course languages that are not statically typed (e.g. Lisp family) or those with restricted use of higher order functions (like Erlang). There are two ways of extending a polymorphic type discipline with subtyping: the simpler one modifies only the typing rules while preserving the syntax of types, the other introduces subtyping constraints into type syntax (thus making instantiation of bound type variables subject to satisfaction of these constraints). Chapters 6 and 7 discuss respective approaches, giving an algebraic characterisation of typability in the first one, and linking the second to subtype inequalities which allows to infer the following conclusion:

There exists a wide class of posets, which contains many important and interesting ones, and for which typability in ML with subtyping has the same complexity as typability without subtyping.

This is the main thesis which we now set out to prove.

[2]This notion is formalized in section 2.5

[3]that is with quantification at top level only

Acknowledgements

This paper would be never written without the continous advice, encouragement and patience of Professor Jerzy Tiuryn. Many thanks go also to Damian Niwiński, Fritz Henglein, Grzegorz Grudziński, Aleksy Schubert and Robert Maron for fruitful discussions and suggestions. I am also grateful to Jakob Rehof and Paweł Urzyczyn for careful lecture of drafts at various stages and helpful remarks.

Part I

Subtype inequalities

Chapter 2

Preliminaries

Contents

2.1 Motivations

The following chapters discuss various aspects of a single decision problem: satisfiability of subtype inequalities[1] (abbreviated SSI). This problem, by itself interesting and presenting numerous challenges, is very closely related to many type reconstruction problems. Some of the connections are presented below; others are unveiled in the second part of the thesis.

Assuming we have already defined a subtype ordering, the simplest way to extend simple-typed lambda-calculus with subtyping is just to add the subsumption rule to the original system:

$$E \cup \{x : \tau\} \vdash x : \tau$$

$$\frac{E \vdash M : \tau \to \rho \quad E \vdash N : \tau}{E \vdash (MN) : \rho}$$

$$\frac{E \cup \{x : \tau\} \vdash M : \rho}{A \vdash (\lambda x.M) : \tau \to \rho}$$

$$\frac{E \vdash M : \tau \quad \vdash \tau \leq \rho}{E \vdash M : \rho}$$

Without the last (subsumption) rule, this system becomes an ordinary system of simple types. Then we say that ρ is an instance of τ ($\rho \preceq \tau$) if there exists a substitution S such that $S\tau = \rho$. But with subsumption rule, it makes sense to extend the notion of instance to all cases where $S\tau \leq \rho$.

In general, we say that a type system enjoys the *principal types property*, if for every typable term there exists a type representing all possible types of this term. Usually this means that for every term there is a type such that all types of the term are instances of the former type.

To see that the principal types property fails for this system (even with an extended notion of instance), let us consider the term

$$\mathbf{2} \equiv \lambda f.\lambda x.f(fx)$$

In the system without subtyping its principal type is (the type of Church numerals)

$$(\alpha \to \alpha) \to (\alpha \to \alpha)$$

Unless the underlying ordering is trivial (i.e. equality), this is not a principal type in the system with subsumption. To see this, assume that our type system includes type constants *int* and *real* with subtyping relation $int \leq real$. Then

$$\vdash \mathbf{2} : (real \to int) \to (real \to int)$$

[1]This problem is formally defined in section 2.2.

But this type is not an instance of $(\alpha \to \alpha) \to (\alpha \to \alpha)$. On the other hand another candidate for a principal type

$$(\alpha \to \beta) \to (\alpha \to \beta)$$

also fails, since

$$\not\vdash \mathbf{2} : (\mathit{int} \to \mathit{real}) \to (\mathit{int} \to \mathit{real})$$

John C. Mitchell in his seminal paper [Mit84] proposed as a subtyping extension of simply-typed λ-calculus a system, where apart from the usual environment binding types to variables he uses also a set of subtyping assumptions. He goes on to show that typability in this system can be reduced to SSI and therefore is decidable.

$$C, E \cup \{x : \sigma\} \vdash_M x : \sigma$$

$$\frac{C, E \vdash_M M : \sigma \to \tau \quad C, E \vdash_M N : \sigma}{C, E \vdash_M (MN) : \tau}$$

$$\frac{C, E \cup \{x : \sigma\} \vdash_M M : \tau}{C, E \vdash_M (\lambda x.M)\sigma \to \tau}$$

$$\frac{C, E \vdash_M M : \sigma \quad C \vdash_M \sigma \leq \tau}{C, E \vdash_M M : \tau}$$

In this system one has to talk about *principal typings* rather than principal types, but we as we shall see later, this idea can be used in other type systems to fully recover principal types property for closed terms.

To illustrate importance of SSI, let us quote the critique of Mitchell himself about this system [LM92]:

> A simple example that illustrates one of the problems with type constants is the signature with type constants *int* and *real*, with $\mathit{int} \leq \mathit{real}$ and term constants $1, 2 : \mathit{int}$, $\mathit{mult} : \mathit{int} \to \mathit{int} \to \mathit{int}$ and $\mathit{div} : \mathit{real} \to \mathit{real} \to \mathit{real}$. [...] consider the expression,
>
> $$\mathit{mult}(\mathit{div}\ 1\ 2)\ 2$$
>
> This is not well-typed, given the signature, since the subexpression (div 1 2) only has type *real* and not type *int*. The typing algorithm in [Mit84, Mit91] [...] *would* produce a typing statement for this term, namely
>
> $$\mathit{real} \leq \mathit{int} \vdash_M (\mathit{mult}(\mathit{div}\ 1\ 2)\ 2) : \mathit{int}$$

Indeed, Hoang and Mitchell [HM95] have shown that typability in this system is equivalent to SSI. The remainder of this chapter presents the latter problem as well as notions and problems pertinent to it.

2.2 Subtype inequalities

Let Q be a finite poset. The elements of Q are constant symbols of the signature which in addition contains a binary operation symbol $\rightarrow$. Let $\mathcal{T}_Q$ be the term algebra over this signature. The carrier of $\mathcal{T}_Q$ is partially ordered by extending the order from Q to all terms by the rule

$$\frac{r_1 \leq t_1 \quad t_2 \leq r_2}{(t_1 \rightarrow t_2) \leq (r_1 \rightarrow r_2)}$$

A *system* Σ of *inequalities* is a finite set of formulas of the form

$$\Sigma = \{\tau_1 \leq \rho_1, \ldots, \tau_n \leq \rho_n\},$$

where τ's and ρ's are terms over the above signature with variables from a set V. Σ is said to be *flat* if every term in Σ is of size 1, i.e. it is either a constant symbol or a variable. Σ is said to be *satisfiable* in $\mathcal{T}_Q$ if there is a valuation $v : V \rightarrow \mathcal{T}_Q$ such that $\tau_i[v] \leq \rho_i[v]$ holds in $\mathcal{T}_Q$ for all i.

Satisfiability of Subtype Inequalities (SSI) is the following problem: given a system of inequalities Σ, decide whether it is satisfiable (the poset Q is considered to be fixed, rather then a part of the problem).

Similarly, FLAT-SSI is the problem of deciding whether given flat system of inequalities is decidable.

2.3 Shapes and weak satisfiability

The set $\mathcal{T}_\star$ of *shapes* is the set of terms without variables over the signature $\Sigma = \langle 0, \rightarrow \rangle$.

We shall use the canonical map $(\cdot)_\star : \mathcal{T}_Q(V) \rightarrow \mathcal{T}_\star(V)$

$$(c)_\star = 0 \text{ for } c \in Q, \qquad (v)_\star = v \text{ for } v \in V \qquad (t \rightarrow u)_\star = (t)_\star \rightarrow (u)_\star$$

and call $(t)_\star$ the shape of t if t is a term without variables.

Note that the subtype order on $\mathcal{T}_Q$ is stratified, i.e. only terms of the same shape are comparable. In the sequel we shall operate on strata of this ordering, defined as follows:

$$\begin{aligned} Q_0 &= Q \\ Q_{\sigma \rightarrow \tau} &= \{t \rightarrow u : t \in Q_\sigma, u \in Q_\tau\} \end{aligned}$$

A system of inequalities $\Sigma = \{\tau_1 \leq \rho_1, \ldots, \tau_n \leq \rho_n\}$ is said to be *weakly satisfiable* if $\Sigma_\star = \{(\tau_1)_\star = (\rho_1)_\star, \ldots, (\tau_n)_\star = (\rho_n)_\star\}$ is satisfiable in $\mathcal{T}_\star$.

Weak satisfiability is clearly a necessary condition for satisfiability. It is decidable in (and in fact complete for) polynomial time since it is an instance of the unification problem [Tiu92, DKM84].

In the sequel, we shall deal only with weakly satisfiable sytems. In some places we shall assume (for the sake of proofs, not algorithms) that all inequalities of the system are annotated with proper shape and use the notation

$$t \leq_\sigma u$$

for an inequality in shape σ.

2.4 Partial orders

A partially ordered set (poset for short) is a set equipped with a reflexive, transitive and antisymmetric relation $\leq$. By a poset dual to $\langle Q, \leq \rangle$ we mean the poset $\langle Q, \leq^{-1} \rangle$.

Whenever there are no doubts which ordering relation on Q is meant, we shall use the notation

$$Q \models a \leq b$$

to express the fact that $a \leq b$ holds in $\langle Q, \leq \rangle$.

2.4.1 Retractions and obstacles

Let Q and R be posets. We say that R extends Q if Q is a subposet of R. We say that R retracts to Q ($R \rhd Q$) if there exists an order preserving and idempotent (i.e. such that $f \circ f = f$) map $f : R \to Q$.

The problem of *Q-retractability* is defined as follows: given $R \supseteq Q$, does R retract to Q?

For every finite poset Q, Q-FLAT-SSI is logspace-equivalent to the problem of Q-retractability. Henceforth we shall identify flat systems of inequalities over Q with corresponding extensions of Q.

V. Pratt and J. Tiuryn [PT96] introduce the notion of an *obstacle* to retractability — a property of a larger poset which prevents it from retracting onto another one. An obstacle is called complete for Q if R retracts to Q whenever R does not satisfy it. In the mentioned paper, they discuss a class of posets (which they call TC-feasible) for which complete obstacles can be expressed by formulae of logic with a transitive closure operator. This subject is explored further in Chapter 4. In Chapter 3 we discuss other kind of obstacles, which we consider useful in connection with inheritance.

The notion of retraction and the retractability problem can be generalized to the case when R is a preorder in an obvious manner. As a flat

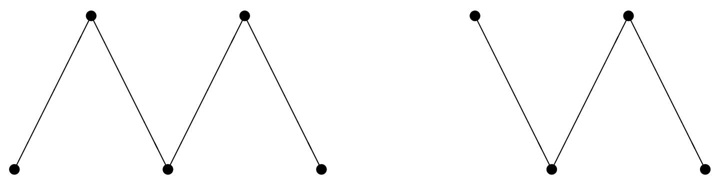


Figure 2.1: Fences: (a) F_4^+ (b) F_3^-

system of inequalities can be naturally viewed as a preorder (modulo transitive closure, that is), we find the preorder formulation more convenient for our application. The obstacles for preorder retraction are the same as for poset retraction.

2.4.2 Distances and disks in a poset

2.4.1 Definition
By an *up-fence* of length n (denoted by F_n^+) from x to y we mean a set $\{x_0, \ldots, x_n\}$, such that $x_0 = x, x_n = y$, and $x_{2i} \leq x_{2i+1} \geq x_{2i+2}$ for all relevant i.

The *up-distance from x to y in Q*, $d^{+1}(x, y)$ is the smallest number n such that there exists an up-fence of length n from x to y. If such n does not exist, we assume $d^{+1}(x, y) = +\infty$.

The notions of a *down-fence* (F_n^-) and *down-distance* ($d^-(x, y)$) are defined dually.

In the sequel, we shall use distances introduced above in the context

$$Q \models d^\varepsilon(x, y) \leq n$$

(with ε being either -1 or $+1$) by which we mean that the respective distance in Q does not exceed n.

2.4.2 Definition
By a *disk in Q* with the center a and radius (r^{+1}, r^{-1}) we shall mean the set

$$D_Q(a, (r^{+1}, r^{-1})) = \{x \in Q : Q \models d^\varepsilon(a, x) \leq r^\varepsilon \quad \varepsilon \in \{-1, 1\}\}$$

2.4.3 Absolute retracts

Nevermann and Rival [NR85] describe a class of posets for which there exists a simple condition which is necessary and sufficient for retractibility, which they call absolute retracts.

2.4.3 Definition
A set

$$\mathcal{H} = \{(v_t, \delta_t^{+1}, \delta_t^{-1}) : t \in T\}$$

where T is an index set, $v_t \in Q$, $\delta_t^{+1}, \delta_t^{-1} \in N$, is a hole in Q if

$$\bigcap_{t \in T} D_Q(v_t, \delta_t^{+1}, \delta_t^{-1}) = \emptyset$$

and $\mathcal{H}$ contains no proper nonempty sub-family which determines a family of disks with empty intersection.

We say that this hole is separated in $P \supseteq Q$ if

$$\bigcap_{t \in T} D_P(v_t, \delta_t^{+1}, \delta_t^{-1}) = \emptyset$$

2.4.4 Definition
A partially ordered set Q is called an absolute retract if it is a retract of any poset in which all its holes are separated.

2.4.4 Helly posets

2.4.5 Definition
A partially ordered set satisfies the the two disk property (alias Helly property) if, for each family $\mathcal{D}$ of disks

$$\bigcap \mathcal{D} \neq \emptyset$$

whenever

$$D_1 \cap D_2 \neq \emptyset$$

for each D_1, D_2 in $\mathcal{D}$.

2.4.6 Example
The following are examples of Helly posets [Qui83, NR85]:

- lattices
- trees
- fences

2.4.7 Proposition ([NR85])
Every Helly poset is an absolute retract. Helly posets are closed under products, retractions and disjoint unions.

2.4.8 Corollary
If Q is a Helly poset, then so is Q_σ for any shape σ.

2.5 Order varieties and representations

Duffus and Rival [DR81] propose a structure theory for ordered sets basing on direct product and retract as canonical constructions.

2.5.1 Notation
The class of all retracts of Q is denoted by $\mathbf{R}(Q)$.

2.5.2 Definition (Order variety)
An *order variety* is a class $\mathcal{K}$ of ordered sets which is closed under the formation of all direct products of nonempty families of members of $\mathcal{K}$ and under the formation of all retracts. A *weak order variety* is a class closed under finite products and retracts.

2.5.3 Definition (Representation)
A family $\{Q_i\}_{i\in I}$ of ordered sets *represents* (or: *is a representation of*) Q if $Q_i \in \mathbf{R}(Q)$ for each $i \in I$ and $Q \in \mathbf{R}(\prod_{i\in I} Q_i)$.

2.5.4 Definition (Irreducibility)
An ordered set Q is *irreducible* if, for each representation $\{Q_i\}_{i\in I}$ of Q, $Q \in R(Q_i)$ for some $i \in I$.

To see that (weak) order varieties are a useful tool in the complexity analysis of the retractability problem (and hence also the SSI problem) let us recall two facts proved in [PT96]:

2.5.5 Proposition
Let Q be a poset which extends $P_1 \times P_2$. Let $a \in P_1$ and $b \in P_2$ be arbitrary elements. Then Q retracts on $P_1 \times P_2$ iff Q retracts on both: $P_1 \times \{b\}$ and $\{a\} \times P_2$.

Let Q and P_1 be extensions of a poset P. We construct a P_1-extension Q^+ as follows. Add to Q the new elements in $P_1 - P$, (without loss of generality we may assume that $P_1 - P$ and Q are disjoint). The order relation $\leq_+$ in Q^+ is defined as follows. $x \leq_+ y$ iff one of the following holds.

- $x \leq y$ holds in Q.
- There exists $z \in P$, such that $x \leq z$ holds in Q and $z \leq y$ holds in P_1.
- There exists $z \in P$, such that $x \leq z$ holds in P_1 and $z \leq y$ holds in Q.
- $x \leq y$ holds in P_1.

Then

2.5.6 Proposition
(i) $\leq_+$ is a partial order in Q^+.
(ii) P^+ is order-isomorphic to P_1.
(iii) If Q retracts to P, then Q^+ retracts to P_1.
(iv) If P_1 retracts to P, then for every poset Q being an extension of P, Q retracts to P, iff Q^+ retracts to P_1.

From these two facts it follows that if $\mathcal{K}$ is a weak order variety, then to prove that the retractability problem is decidable in polynomial time for all members of $\mathcal{K}$, it is sufficient to prove this fact for irreducibles representing $\mathcal{K}$.

The second argument for importance of order varieties in complexity analysis of SSI lies in the very definition of the problem: from the definition of ordering on arrow types it follows that the class of posets for which we want to prove tractability of SSI must be closed under finite products.

We adopt this approach in Chapter 3, basing on the following fact, proved in [DR81]:

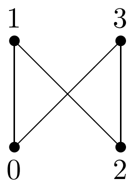
2.5.7 Proposition
Every finite fence is irreducible. Moreover, the class of finite Helly posets is an order variety generated by all finite fences.

An attempt to apply the same method to absolute retracts fails: even though their class is an order variety, no characterisation of irreducibles in this class is known. However, as observed in [PT96], for every absolute retract, a necessary and sufficient condition for retractability (or a complete obstacle) can be expressed by a formula of transitive closure logic, therefore each absolute retract is TC-feasible. In Chapter 4 we extend the TC logic of Pratt and Tiuryn to cater for subtype inequalities design an inference system for this logic and by analysing its properties finally come to conclusion that SSI can be decided in polynomial time also for TC-feasible posets.

2.6 Intractable posets

An n-crown is a poset with $2n$ elements $0, 1, \ldots, 2n-1$ ordered in such a way that $2i \leq (2i \pm 1) \bmod 2n$.

V. Pratt and J. Tiuryn [PT96] show that for n-crowns ($n \geq 2$), FLAT-SSI is NP-complete. Moreover, in [Tiu92] it is shown that for these posets SSI is PSPACE-hard. In Chapter 5 we show how this result can be generalized.



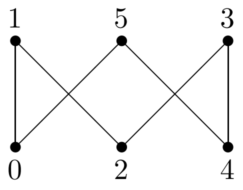


Figure 2.2: (a) 2-crown (b) 3-crown

2.7 Complexity classes

In naming complexity classes we generally follow the conventions of [BDG95, BDG90]. By $DTM(s,t)$ we understand a class of problems decidable by an s-space bounded and t-time bounded deterministic Turing machine. Expressions $NTM(s,t)$ and $ATM(s,t)$ denote corresponding nondeterministic and alternating classes.

Beside obvious complexity classes, we use the following abbreviations (ATIME and ASPACE denote alternating time and space respectively, cf. e.g. [BDG95, Pap94]):

$$NLOGSPACE = NSPACE(\log n) \tag{2.1}$$
$$ALOGSPACE = ASPACE(\log n) \tag{2.2}$$
$$AP = \bigcup_{c>0} ATIME(n^c) \tag{2.3}$$

The correspondence between alternating and deterministic complexity classes is established by the following

$$ASPACE(s(n)) = \bigcup_{c>0} DTIME(c^{s(n)}) \tag{2.4}$$
$$ATIME(t(n)) \subseteq DSPACE(t^2(n)) \tag{2.5}$$

in particular, we have

$$ALOGSPACE = P \tag{2.6}$$
$$AP = PSPACE \tag{2.7}$$

Chapter 3

Trees and inheritance

Contents

3.1 Trees, inheritance and Helly posets

Why trees are important The original motivation for studying orders being trees was that trees correspond to the coercion structure of single-inheritance object systems. Although traditionally, recursive types are used for inheritance mechanism, recent works show that it is possible to construct type systems for inheritance which do not rely on recursive types (cf. e.g. [PT92, PT93]).

On the other hand trees are present even if we consider simple coercion. Perhaps the simplest possible example is a numeric or enumeration type and a couple of its subrange types.

Example One may consider types (like in the language C) *signed int* $= -2^{15}..2^{15}-1$, *unsigned int* $= 0..2^{16}-1$, *long int* $= -2^{31}..2^{31}-1$ and coercions between them based on respective inclusions. Then the poset would look like

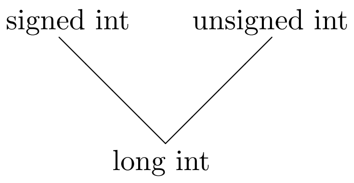


Figure 3.1: An example subtyping relation involving subrange types

Why trees aren't trivial The main problem is that since we consider non-flat inequalities, any result for trees actually has to cover a wider range of posets, since any algorithm working for a given poset, must obviously work for the poset formed of terms of a given shape over this poset. Therefore the problem is in finding a class of posets wide enough to facilitate inductive proof, and narrow enough to fulfill satisfactory thesis.

Example Consider set of types from previous exmaple

$$Q = \{\textit{signed int, unsigned int, long}\}$$

and let

$$Q_{0\to 0} = \{\rho \to \tau : \rho, \tau \in Q\}$$

then inequalities between elements of $Q_{0\to 0}$ form a poset depicted in Fig. 3.2. This poset seems all but trivial. In fact it bears some resemblance to the

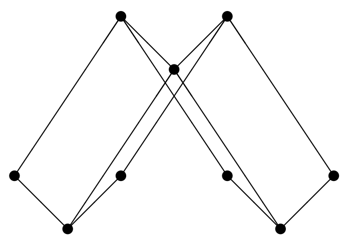


Figure 3.2: The poset $Q_{0\to 0}$

posets for which the type reconstruction is NP-hard [LM92] and the SSI problem is PSPACE-hard [Tiu92] — crowns (discussed in section 2.6).

In fact, because of this resemblance it had been for some time conjectured that the SSI problem for trees is NP-hard. However, as we prove in this chapter, this is not the case and said resemblance turns out to have been deceiving.

The key to solving SSI problem for trees is the class of Helly posets, introduced in section 2.4.4. It includes trees and is closed under retractions, products, duality and disjoint unions. In this chapter we prove that for Helly posets, SSI is decidable in P and FLAT-SSI in NLOGSPACE.

3.2 Inference systems

In this section we are going to present inference systems which for a given system of inequalities Σ, allow to infer formulae that must be satisfied by any solution of Σ (i.e. are sound). In general, they are not complete, since they have been designed for efficient derivability checking. On the other hand, some weak (but sufficient for our purposes) completeness results are proven in the next section.

3.2.1 Ground consistency

Following [Tiu92], we introduce a system which allows to infer judgements of the form $\Sigma \vdash_s t_1 \leq t_2$, where t_1, t_2 are subterms of terms occurring in Σ:

$$\Sigma \vdash_s \tau \leq \rho \quad \text{for } \tau \leq \rho \in \Sigma \tag{3.1}$$

$$\frac{\Sigma \vdash_s \tau \leq \rho \quad \Sigma \vdash_s \rho \leq \sigma}{\Sigma \vdash_s \tau \leq \sigma} \tag{3.2}$$

$$\frac{\Sigma \vdash_s (\tau_1 \to \tau_2) \leq (\rho_1 \to \rho_2)}{\Sigma \vdash_s \rho_1 \leq \tau_1, \quad \tau_2 \leq \rho_2} \tag{3.3}$$

3.2.1 Definition
We call Σ *ground consistent* whenever for each $c_1, c_2 \in Q$ if $\Sigma \vdash_s c_1 \leq c_2$, then $Q \models c_1 \leq c_2$.

3.2.2 Theorem
Ground consistency is complete for NLOGSPACE with respect to NC1 reductions

Proof: First we are going to prove that the problem of checking ground consistency is co-NLOGSPACE hard. Since nondeterministic space is closed under complementation [Imm88], it follows that checking ground consistency is NLOGSPACE-hard.

The proof goes by reduction of the reachability in directed graph. Given a graph $G = \langle V, E \rangle$, and a pair of vertices $a, b \in V$, take $Q = \{a, b\},\ a \leq b$. All other vertices are treated as constants. For each $(x, y) \in E$ put an inequality $x \leq y$ in Σ. Thus defined, Σ is ground consistent iff there is no path from b to a in G.

Note that the reduction described above can be in fact carried out in NC1. Thus ground consistency is NLOGSPACE-hard with regard to NC1 reductions.

To see that ground consistency is indeed in NLOGSPACE consider the algorithm which we exhibit in more detail for the flat case:

- nondeterministically choose a pair of constants a, b such that $a \leq b$ does not hold in Q
- check that $\Sigma \vdash_s a \leq b$ by nondeterministically choosing inequalities from Σ that form a chain from a to b:

  $$a \leq \alpha_1,\ \alpha_1 \leq \alpha_2, \ldots, \alpha_n \leq b,$$

  where $\alpha_i \in Q \cup V$ for $i = 1, \ldots, n$

Indeed, logarithmic space is sufficient to record consecutive α's, since we only need to record the current element, rather than the whole chain.

To extend this algorithm for general case, we should systematically number subterms of Σ, then guess the pair of constants and nondeterministically detect the inference that leads to an inconsistency. ∎

3.2.2 Distance consistency

The next system we present (an indeed a central one for this section), allows to infer judgements about distances. It is easy to prove that the system is sound; cf. figures 3.4 through 3.7 for intuitions behind the rules.

$$\text{[Ax]} \qquad \Sigma \vdash_d d^{+1}(x,y) \leq 1 \quad \text{for } x \leq y \in \Sigma$$

$$\text{[Symm]} \qquad \frac{\Sigma \vdash_d d^{\varepsilon}(x,y) \leq n}{\Sigma \vdash_d d^{(\varepsilon(-1)^n)}(y,x) \leq n}$$

$$\text{[Dual]} \qquad \frac{\Sigma \vdash_d d^{\varepsilon}(x,y) \leq n}{\Sigma \vdash_d d^{(-\varepsilon)}(x,y) \leq n+1}$$

$$\text{[Join]} \qquad \frac{\Sigma \vdash_d d^{\varepsilon}(x,y) \leq n,\ d^{\varepsilon(-1)^{n-1}}(y,z) \leq k}{\Sigma \vdash_d d^{\varepsilon}(x,z) \leq n-1+k}$$

$$\text{[Subterm]} \qquad \frac{\Sigma \vdash_d d^{\varepsilon}(x_1 \to x_2, y_1 \to y_2) \leq n}{\Sigma \vdash_d d^{-\varepsilon}(x_1,y_1) \leq n, \Sigma \vdash_d d^{\varepsilon}(x_2,y_2) \leq n}$$

Figure 3.3: A system for deriving distance judgements.

3.2.3 Definition
A system of inequalities Σ is called *distance consistent* whenever for each $c_1, c_2 \in Q$ and $n \in N$ if $\Sigma \vdash_d d^{\varepsilon}(c_1,c_2) \leq n$, then $Q \models d^{\varepsilon}(c_1,c_2) \leq n$.

Distance consistency is decidable in polynomial time. We can construct two $O(|\Sigma|) \times O(|\Sigma|)$ arrays, for d^{+1} and d^{-1} respectively in such a way that in the array for d^{ε}, in the cell indexed by τ and ρ there is the least number n such that $\Sigma \vdash_d d^{\varepsilon}(\tau,\rho) \leq n$ holds, or (a special symbol for) $+\infty$ if no such n exists.

In fact, even the following holds:

3.2.4 Theorem
Distance consistency is NLOGSPACE complete.

Proof: Hardness is obvious, since ground consistency is but a special case of distance consistency. On the other hand, an algorithm to check the latter can be constructed easily, following the one described in the proof of Theorem 3.2.2. The only difference is that the direction of inequalities may vary throughout the chain; we must maintain a counter for the number of such alternations. Only constant space is needed for this counter, since maximum distance is fixed with the given poset, and we are looking for fences *shorter* than given distance. ∎

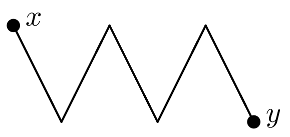


Figure 3.4: An example for the [Symm] rule: $d^{+1}(y, x) \leq 5$, $d^{-1}(x, y) \leq 5$.

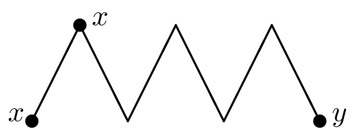


Figure 3.5: An example for the [Dual] rule: $d^{-1}(x, y) \leq 5$, but since $x \leq x$, we have $d^{+1}(x, y) \leq 6$.

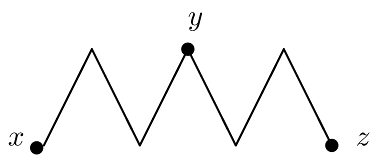


Figure 3.6: An example for the [Join] rule: $d^{+1}(x, y) = 3$, $d^{-1}(y, z) = 3$, joined they yield $d^{+1}(x, z) = 6$.

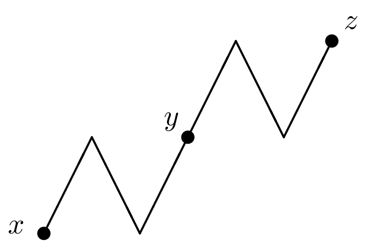


Figure 3.7: Another example for the [Join] rule: $d^{+1}(x, y) \leq 3$, $d^{+1}(y, z) \leq 3$, yield when joined $d^{+1}(x, z) \leq 5$.

3.3 Completeness results

The main result presented in this chapter is that the inference system $\vdash_d$ is weakly complete in the sense that every weakly satisfiable system of inequalities over a Helly poset is satisfiable iff it is distance consistent. Thus satisfiability of such systems is decidable in polynomial time.

3.3.1 Lemma

(a) $\Sigma \vdash_s x \leq y$ *iff* $\Sigma \vdash_d d^{+1}(x, y) \leq 1$

(b) $\Sigma \vdash_s y \leq x$ *iff* $\Sigma \vdash_d d^{-1}(x, y) \leq 1$

Proof: In the proof of (a), the "only if" part follows from axiom 3.4. On the other hand, the "if" part can be proved by simple induction on the derivation of $\Sigma \vdash_d d^{+1}(x, y) \leq 1$. For the proof of (b) we can use the result of (a) and the rule [Symm]:

$$\frac{\Sigma \vdash_d d^{+1}(y, x)}{\Sigma \vdash_d d^{-1}(x, y)}$$

∎

3.3.2 Corollary

If Σ is distance consistent then it is also ground consistent.

3.3.1 Flat systems and retractability

Consider a flat and ground consistent system of inequalities Σ. Let us define the following equivalence relation $\simeq$ on $var(\Sigma)$:

$$x \simeq y \iff \Sigma \vdash_s x \leq y \wedge \Sigma \vdash_s y \leq x$$

By an *extension of Q generated by* Σ we shall mean the set

$$Q_\Sigma = Q \cup var(\Sigma)/\simeq$$

with the order defined as the transitive closure of the sum of the order on Q and the order on $Q \cup var(\Sigma)/\simeq$, induced by $\Sigma \vdash_s$.

3.3.3 Lemma

A flat system Σ of inequalities over Q is satisfiable in $\mathcal{T}_Q$ iff it is satisfiable in Q.

Proof: The "if" part is obvious. For the proof of the "only if" part, let $\sigma : V \to \mathcal{T}_Q$ be a solution of Σ and c be an arbitrary element of Q. Consider $\sigma' : V \to Q$ defined as follows:

$$\sigma'(v) = \begin{cases} \sigma(v) & \text{if } \sigma(v) \in Q \\ c & \textit{otherwise} \end{cases}$$

It is easy to see that σ' is indeed a solution of Σ. ∎

3.3.4 Lemma
A flat system of inequalities Σ is satisfiable if and only if Σ is ground consistent and $Q_\Sigma \rhd Q$.

Proof: Each retraction $f : Q_\Sigma \to Q$ determines a solution of Σ:

$$\sigma(x) = f([x]_\simeq)$$

On the other hand, if σ is a solution of Σ, then

$$x \simeq y \Rightarrow \sigma(x) = \sigma(y)$$

∎

3.3.5 Lemma
Let Σ — ground consistent. Then:

(a) *If $Q_\Sigma \models [x]_\simeq \leq c$ for some $x \in var(\Sigma)$, $c \in Q$ then there exists $c' \in Q$ such that*
$$\Sigma \vdash_s x \leq c' \quad Q \models c' \leq c$$

(b) *If $Q_\Sigma \models c \leq [x]_\simeq$ for some $x \in var(\Sigma)$, $c \in Q$ then there exists $c' \in Q$ such that*
$$\Sigma \vdash_s c' \leq x \quad Q \models c \leq c'$$

(c) *If $Q_\Sigma \models [x_1]_\simeq \leq [x_2]_\simeq$ for some $x_1, x_2 \in var(\Sigma)$ then either*
$$\Sigma \vdash_s x_1 \leq x_2$$
or there exist $c_1, c_2 \in Q$ such that
$$\Sigma \vdash_s x_1 \leq c_1 \quad Q \models c_1 \leq c_2 \quad \Sigma \vdash_s c_2 \leq x_2$$

Proof: (a) Since order on Q_Σ is the transitive closure of the sum of order on Q and the one induced by $\Sigma \vdash_s$, there exist $p_0, p_1, \ldots, p_n$ such that $p_0 = [x]$, $p_n = c$ and for every $1 \leq i \leq n$ one of the following holds:

(i) $Q \models p_{i-1} \leq p_i$, or

(ii) $p_{i-1} = [w_{i-1}],\ p_i = [w_i],\ \Sigma \vdash_s w_{i-1} \leq w_i$, or

(iii) $p_{i-1} = [w_{i-1}],\ p_i \in Q,\ \Sigma \vdash_s w_{i-1} \leq p_i$, or

(iv) $p_{i-1} \in Q,\ p_i = [w_i],\ \Sigma \vdash_s p_{i-1} \leq w_i$.

Let k be the smallest such that $p_k \in Q$. From ground consistency of Σ it follows that $p_k \leq c$ holds in Q. On the other hand for all $0 \leq i \leq k-1$, $p_i = [w_i]$ for some $w_i \in var(\Sigma)$ and thus by the rule 3.2 we have $\Sigma \vdash_s x \leq p_k$.

The proof of (b) is analogous, and to prove (c) we should use the same argument twice. It is worth mentioning that we cannot get rid of the second part of the alternative in (c), i.e. it is not always true that if

$$Q_\Sigma \models [x_1] \leq [x_2]$$

then

$$\Sigma \vdash_s x_1 \leq x_2.$$

∎

3.3.6 Lemma
If Σ is distance consistent than we have for any $c_1, c_2 \in Q$

$$Q_\Sigma \models d^\varepsilon(c_1, c_2) \leq n \iff Q \models d^\varepsilon(c_1, c_2) \leq n$$

Proof: The "if" part is obvious, since Q_Σ is an extension of Q.

The proof of the "only if" part is by induction on n (simultaneously for $\varepsilon = +1$ and -1). The basis of induction follows from the lemma 3.3.5.

Now, for the induction step: let $c_1 = p_0, p_1, \ldots, p_n = c_2$ be a fence in Q_Σ, connecting c_1 with c_2. If $p_i \in Q$ for some i, $0 < i < n$, then we have

$$Q_\Sigma \models d^\varepsilon(c_1, p_i) \leq i$$

$$Q_\Sigma \models d^{\varepsilon(-1)^i}(p_i, c_2) \leq n - i$$

so that we may use the induction hypothesis (since both i and $n-i$ are less than n) to obtain

$$Q \models d^\varepsilon(c_1, p_i) \leq i$$

$$Q \models d^{\varepsilon(-1)^i}(p_i, c_2) \leq n - i$$

from which the thesis follows.

On the other hand, if none of $p_1, \ldots, p_n - 1$ belong to Q, then by definition of ordering on Q_Σ and of relation $\simeq$ we have that

$$\Sigma \vdash_d d^\varepsilon(c_1, c_2) \leq n$$

and the thesis follows from the distance consistency of Σ.

∎

3.3.7 Theorem
A flat system of inequalities over a poset satisfying the Helly property is satisfiable if and only if it is distance consistent.

Proof: It is easy to see that distance consistency is indeed a necessary condition for satisfiability. In order to prove that it is also sufficient, we shall show that $Q_\Sigma \rhd Q$ and use Lemma 3.3.4. Since Q satisfies the Helly property and thus is an absolute retract, it is sufficient to prove that all holes in Q of cardinality 2 are separated in Q_Σ.

Let us assume the contrary, i.e. that there exists a pair of disks which is a hole in Q that is not separated in Q_Σ. This implies that we may chose $a_1, a_2 \in Q,\ r_1^{+1}, r_1^{-1}, r_2^{+1}, r_2^{-1} \in N,\ x \in Q_\Sigma$ such that

$$D_Q(a_1, r_1^{+1}, r_1^{-1}) \cap D_Q(a_2, r_2^{+1}, r_2^{-1}) = \emptyset$$

$$x \in D_{Q_\Sigma}(a_1, r_1^{+1}, r_1^{-1}) \cap D_{Q_\Sigma}(a_2, r_2^{+1}, r_2^{-1})$$

We can further assume that the sum $r_1^{+1} + r_1^{-1} + r_2^{+1} + r_2^{-1}$ is minimal (i.e. there is no pair of disks fulfilling above conditions but with lower sum of radii).

If $x \in Q$, we can use Lemma 3.3.6 to obtain contradiction with distance consitency of Σ. Otherwise $x = [v]_\simeq$ for some $v \in var(\Sigma)$. Now, if

$$\Sigma \vdash_d d^\varepsilon(a_1, v) \leq r_1^\varepsilon$$

and

$$\Sigma \vdash_d d^\theta(a_2, v) \leq r_2^\theta$$

then by the rules [Join] and [Symm] we can infer that

$$\Sigma \vdash_d d^\varepsilon(a_1, a_2) \leq r$$

while $Q \not\models d^\varepsilon(a_1, a_2) \leq r$, where $r = r_1^\varepsilon + r_2^\theta - 1$ if $(-1)^{r_1^\varepsilon} = (-1)^{r_1^\theta}$, and $r = r_1^\varepsilon + r_2^\theta$ otherwise. That contradicts the distance consistency of Σ.

If the opposite holds (i.e. $\Sigma \not\vdash_d d^\varepsilon(a_1, v) \leq r_1^\varepsilon$ or $\Sigma \not\vdash_d d^\theta(a_2, v) \leq r_2^\theta$), we can assume without loss of generality that

$$\Sigma \not\vdash_d d^\varepsilon(a_1, v) \leq r_1^\varepsilon$$

Let $a_1 = x_0, x_1, \ldots, x_n = [v]$ be a fence in Q_Σ, connecting a_1 with $[v]_\simeq$, and i be the largest integer such that

$$\Sigma \not\vdash_d d^\varepsilon(x_i, v) \leq r_1^\varepsilon - i$$

(note that $i < r_1^\varepsilon$). In other words we have (for $\alpha = \varepsilon(-1)^{i+1}$)

$$\Sigma \vdash_d d^\alpha(x_{i+1}, v) \leq r_1^\varepsilon - i - 1$$

$$\Sigma \not\vdash d^{-\alpha}(x_i, x_{i+1}) \leq 1$$

$$Q_\Sigma \models d^{-\alpha}(x_i, x_{i+1}) \leq 1$$

If $i = 0$, then by Lemma 3.3.5 there exists $a' \in Q$ such that

$$Q \models d^{\varepsilon}(a_1, a') \leq 1$$

and

$$\Sigma \vdash_d d^{\varepsilon}(a', x_1) \leq 1$$

Thus we can consider a' instead of a_1, and $i = 1$ instead of 0.

Neither x_i nor x_{i+1} can be a constant from Q, since then we could place the center of disk in x_i, thus obtaining a pair of disks, fulfilling assumptions but with smaller sum of radii, which would contradict the assumption of minimality. But if $x_{i+1} = [w]_{\simeq}$ for some $v \in var(\Sigma)$, then by Lemma 3.3.5 there exists constant c such that

$$\Sigma \vdash_d d^{\alpha}(c, w) \leq 1$$

which again leads to a contradiction with minimality. ∎

3.3.8 Corollary
For Helly posets, FLAT-SSI is NLOGSPACE complete.

Proof: By Theorem 3.2.4 distance consistency is NLOGSPACE-complete. By Theorem 3.3.7 satisfisbility of flat systems of inequalities (i.e. FLAT-SSI) is equivalent to distance consistency. ∎

3.3.2 From FLAT-SSI to SSI

The thesis of Lemma 3.3.7 can be lifted to arbitrary systems of inequalities leading us to the following

3.3.9 Theorem
A system of inequalities Σ over a poset satisfying the Helly property is satisfiable iff it is weakly satisfiable and distance consistent.

Before we go on with the proof of this theorem, let us observe that it generalizes one contained in [Tiu92], covering a much wider range of posets. Since the main work has been done while proving the result for the flat case, now we can follow the proof in [Tiu92] with minor changes only. The most important difference is that we must prove distance consistency instead of ground consistency in the induction hypothesis.

It is also worth noting, that another proof of our theorem can be derived from the proof of Theorem 4.2.16 in the next chapter.

Proof: The "only if" part is obvious. The opposite implication is proved by induction on the number of equivalence classes of $\sim$ defined on $var(\Sigma)$ as follows

$$x \sim y \quad \text{iff} \quad \Sigma_\star \models x = y$$

Suppose first that the quotient set $var(\Sigma)/_\sim$ has only one element. Then we consider the set

$$\hat{\Sigma} = \{\xi \leq \xi' : \Sigma \vdash_s \xi \leq \xi' \text{ and } |\xi| = 1 \text{ or } |\xi'| = 1\}$$

($|\xi|$ denotes size of the term ξ; thus $|\xi| = 1$ means that ξ is either a constant or a variable).

It is easy to prove that Σ is satisfiable iff $\hat{\Sigma}$ is satisfiable. In fact above two systems have the same solutions. It follows that if τ is a term of depth greater than 1 which occurs in $\hat{\Sigma}$ then it must be a term without variables. Thus $\hat{\Sigma}$ can be viewed as a flat system of inequalities. Since Σ is weakly satisfiable and distance consistent it follows that $\hat{\Sigma}$ is too. Hence it has a solution and so has Σ.

Now let us assume that $var(\Sigma)/_\sim$ has $n+1$ elements and let $y \in var(\Sigma)$ be such that for no $z \in var(\Sigma)$ there is a term τ with $|\tau| > 1$, $z \in var(\tau)$ and $\sigma_\star \models \tau = y$. If there is no such y, then one easily finds a term τ such that $|\tau| > 1$, $z \in var(\tau)$ and $\sigma_\star \models \tau = z$. This would contradict weak satisfiability of Σ.

Let

$$[y] = \{z \in var(\Sigma) | z \sim y\} = \{y_1, \ldots, y_k\}$$

and consider the set

$$\hat{\Sigma} = \{\xi \leq \xi' : \Sigma \vdash_s \xi \leq \xi' \text{ and } \xi \in [y] \text{ or } \xi' \in [y]\}$$

It follows from the choice of y that if τ is a non variable term which occurs in $\hat{\Sigma}$, then it contains no variables. Moreover all such terms must be of the same shape, say σ. If there are no such terms then we choose shape σ arbitrarily.

It follows that $\hat{\Sigma}$ can be viewed as a flat system of inequalities over $\mathcal{T}_\sigma$ which satisfies the Helly property. It is weakly satisfiable and distance consistent. Let $\hat{v} : [y] \to \mathcal{T}_\sigma$ be a solution of $\hat{\Sigma}$ and let

$$\Sigma_1 = \hat{v}(\Sigma) = \{\hat{v}(\tau) \leq \hat{v}(\rho) : \tau \leq \rho \in \Sigma\}$$

In the above definition $\hat{v}$ acts as an identity on variables other than those in $[y]$. Let $\sim_1$ be the equivalence relation associated with Σ_1. We claim

$$|var(\Sigma_1)/_{\sim_1}| = n$$

$$\Sigma_1 \text{ is weakly satisfiable}$$

These are easy to prove since $(\Sigma_1)_* = \Sigma_*[\sigma/y_1, \ldots, \sigma/y_k]$, and if $(\Sigma_1)_* \models \tau = \rho$, then $\Sigma_* \models \tau = \rho$.

To complete the proof we need to prove that

$$\Sigma_1 \text{ is distance consistent}$$

Let τ be a term such that it contains no constants, every variable of τ occurs exactly once, $var(\tau) \cap var(\Sigma) = \emptyset$, and for some function $\tilde{v} : var(\tau) \to \{*_1, \ldots, *_m\}$, $\tilde{v}(\tau) = \sigma$. Let $\tau_1, \ldots, \tau_k$ be terms obtained from τ by renaming its variables so that they have pairwise disjoint variables and disjoint from $var(\Sigma)$. Let $\Sigma_2 = \Sigma[\tau_1/y_1, \ldots, \tau_k/y_k]$ and let $\eta : \bigcup_{i=1}^{k} var(\tau_i) \to Q$ be a function such that $\eta(\tau_i) = \hat{v}(y_i)$, for $i = 1, \ldots, k$. Such a function is uniquely determined by the above conditions.

In order to show distance consistency of Σ_1 we need one more definition.

3.3.10 Definition
A sequence $\xi_1, \ldots, \xi_{2\ell}$ of terms in $\mathcal{T}_C(X)$ is called an *η-chain* (of fences) if the following conditions hold

$$\Sigma_2 \vdash_d \; d^{\varepsilon_i}(\xi_i, \xi_{i+1}) \leq n_i, \quad \text{for even } i < 2\ell \tag{3.4}$$

$$\eta(\xi_i) = \eta(\xi_{i+1}), \quad \text{for odd } i < 2\ell \tag{3.5}$$

ℓ is called the *size* of the chain, while its length is computed according to the rule [Join], i.e. adding the lengths of its links and subtracting the number of joints where no direction alteration occurs (cf. figures 3.6 and 3.7

Equality in (3.5) represents syntactic identity of terms. Again, we assume that in the above formula η acts as identity on variables other than those in $\bigcup_{i=1}^{k} var(\tau_i)$. Now, distance consistency of Σ_1 follows immediately from the following two claims.

$$\begin{gathered}\text{For all terms } \tau, \rho, \text{ if } \Sigma_1 \vdash_d d^{\varepsilon}(\tau, \rho) \leq n, \text{ then there} \\ \text{exists an } \eta\text{-chain of length } n \text{ from } \tau \text{ to } \rho \text{ with } \varepsilon_2 = \varepsilon\end{gathered} \tag{3.6}$$

$$\begin{gathered}\text{For all constants } a, b \in Q, \text{ if there is an } \eta\text{-chain of} \\ \text{length } n, \text{ with } \varepsilon_2 = \varepsilon \text{ from } a \text{ to } b, \text{ then } Q \models d^{\varepsilon}(a, b) \leq n\end{gathered} \tag{3.7}$$

The proof of (3.7) is by an obvious induction on the size of η-chain. We sketch the proof of (3.6). We use induction on the length of derivation of $d^{\varepsilon}(\tau, \rho) \leq n$ from Σ_1.

If $\tau \leq \rho$ is in Σ_1, then there is an inequality $\tau' \leq \rho'$ in Σ_2 such that $\eta(\tau') = \tau$ and $\eta(\rho') = \rho$. Thus τ, τ', ρ', ρ is an η-chain with both length and size of 1.

The η-chain for the rule [Symm] is obtained simply by chasing the chain from its premise backwards. For the [Dual] rule we simply apply the same rule to the first chain link in η-chain, obtaining dual η-chain of length greater by one than the original one:

$$\Sigma_2 \vdash_d \ d^{\varepsilon_2}(\xi_2, \xi_3) \leq n_2$$

then by [Dual]

$$\Sigma_2 \vdash_d \ d^{-\varepsilon_2}(\xi_2, \xi_3) \leq n_2 + 1$$

If the last rule in the derivation of $\Sigma_1 \vdash_d d^{\varepsilon}(\tau, \rho) \leq n$ was [Join], then we have $\Sigma_1 \vdash_d d^{\varepsilon}(\tau, \xi) \leq n$ and $\Sigma_1 \vdash_d d^{\varepsilon(-1)^{n-1}}(\xi, \rho) \leq k$ We have two η-chains: one from τ to ξ and another from ξ to ρ. Their concatenation yields an η-chain from τ to ρ. Its length is as stated in the [Join] rule.

Finally, if the last rule was [Subterm], then $\Sigma_1 \vdash_d d^{\varepsilon}(\tau, \rho) \leq n$ must have been obtained from, say $\Sigma_1 \vdash_d d^{\theta}(\xi, \xi') \leq n$. Let $\xi_1, \ldots, \xi_{2\ell}$ be an η-chain of length n from ξ to ξ' and assume that it is one of minimal size. If all terms in that chain are of depth greater than 1, then by applying the rule [Subterm] throughout the chain of fences we get an η-fence from τ to ρ. Otherwise, let ξ_j be a term of depth 1. It cannot be a constant nor a variable in $\bigcup_{i=1}^{k} var(\tau_i)$ since $|\xi| > 1$. Hence $\xi_j \in var(\Sigma) - [y]$. If j is odd then since $|\xi| > 1$ and $|\xi'| > 1$, it follows that $1 < j < 2\ell - 1$ and

$$\Sigma_2 \vdash_d \ d^{\varepsilon_{j-1}}(\xi_{j-1}, \xi_j) \leq n_i, \quad \text{and} \quad \Sigma_2 \vdash_d \ d^{\varepsilon_{j+1}}(\xi_{j+1}, \xi_{j+2}) \leq n_i$$

Moreover, $\eta(\xi_j) = \eta(\xi_{j+1})$. Thus ξ_j and ξ_{j+1} are identical variables and we obtain a contradiction since the η-chain can be shortened (by removing one η-link). The case when j is even is argued similarly. This proves that all terms in the η-chain must be of depth greater than 1, thereby completing the proof of (3.6) and hence the whole proof. ∎

3.3.11 Corollary

For any Helly poset Q and system Σ of inequalities over Q one can check whether Σ is satisfiable in time polynomial with respect to $|\Sigma|$.

Proof: By Theorem 3.3.9, a system of inequalities Σ over a poset satisfying the Helly property is satisfiable iff it is weakly satisfiable and distance consistent. Weak satisfiability is an instance of the unification problem and hence can be checked in polynomial time. On the other hand, from Theorem 3.2.4 it follows that distance consistency can be checked in nondeterministic logarithmic space, hence also in polynomial time. ∎

Chapter 4

Feasible posets

Contents

When dealing with a poset class for which no set of generators is known, order-theoretic notions such as applied in the previous chapter are of less use. Therefore in this chapter we extend another tool which proved useful before — inference systems.

In doing so, we draw upon the work of Pratt and Tiuryn [PT96], who observed that logic with existential quantification and transitive closure operator is useful for describing retractability condidtions for partial orders. On the other hand, validity of formulae of this logic can be checked in NLOGSPACE. In this chapter we show how this idea can be extended to yield a polynomial time algorithm deciding SSI for posets whose retractability conditions can be expressed in the logic mentioned above.

4.1 Annotated TC-formulae

In this section we introduce a variant of logic with transitive closure operator. Syntactically, the main difference from the logic proposed in [PT96] is that since the models we work with are stratified according to shapes, in our logic variables are annotated with shapes.

4.1.1 Syntax

Let Q ba a finite poset, X a set of variables and $\sigma, \sigma_1, \sigma_2 \ldots$ be shapes. First we define the set of σ-shaped terms over Q with variables from a set X, as the smallest set $\mathcal{T}_Q^\sigma(X)$ satisfying the following conditions:

- if $x \in X$ then $x^\sigma \in \mathcal{T}_Q^\sigma(X)$,
- if $q \in Q$ then $q \in \mathcal{T}_Q^0(X)$,
- if $t_1 \in \mathcal{T}_Q^{\sigma_1}(X)$ and $t_2 \in \mathcal{T}_Q^{\sigma_2}(X)$ then $t_1 \to t_2 \in \mathcal{T}_Q^{\sigma_1 \to \sigma_2}(X)$.

Usually we will assume that Q and X is fixed and use a shorthand $t : \sigma$ to mean that t is a term of shape σ.

The set of *annotated TC-formulae over* Q (or, short: ATC-formulae) is the least set ATC_Q such that

- Every atomic formula $t \leq_\sigma u$, where $t, u : \sigma$, is in ATC_Q.
- If φ and ψ are in ATC_Q, and every variable x free in φ and ψ has identical annotations in both formulae, then

  $$(\varphi \vee \psi),\ (\varphi \wedge \psi)$$

  are in ATC_Q.

- If φ is in ATC_Q, and every free occurrence of x is annotated by σ then
$$(\exists x^{\sigma}.\varphi)$$
is in ATC_Q.
- if φ is in ATC_Q, $\vec{\sigma} = \sigma_1, \ldots, \sigma_n$, then
$$TC(\lambda\, \vec{x}^{\vec{\sigma}}, \vec{y}^{\vec{\sigma}}.\varphi)(\vec{t}, \vec{u})$$
is in ATC_Q, where $\vec{x}, \vec{y}$ are n-vectors of individual variables, $\vec{t}, \vec{u}$ are n-vectors of terms such that $t_i, u_i : \sigma_i$.

We shall say that a formula is *flat* if it contains only 0-shaped terms and all its bound variables are annotated with 0. In such a case the annotations are of no consequence and we can safely omit them.

A formula will be called *balanced* if every inequality in it is in the same shape. From now on, we shall deal only with balanced formulae.

4.1.2 Free Variables

Given an ATC-formula φ (or a term t), the set of its *free variables*, $FV(\varphi)$ is defined as usual. It should be stressed that λ in the TC operator is also a binder, so that
$$FV(TC(\lambda\, \vec{x}^{\vec{\sigma}}, \vec{y}^{\vec{\sigma}}.\varphi)(\vec{t}, \vec{u}) = (FV(\varphi) \setminus \{\vec{x}, \vec{y}\}) \cup FV(\vec{t}) \cup FV(\vec{u})$$

4.1.3 Lonely Variables

An occurrence of a variable shall be called *lonely* in φ, if it is free and not inside a term. Formally, given an ATC-formula φ (or a term t), we define the set of its *lonely variables*, $LV(\varphi)$ as follows:
$$\begin{array}{rcl}
LV(x) & = & \{x\} \\
LV(t \to u) & = & \emptyset \\
LV(t \leq u) & = & LV(t) \cup LV(u) \\
LV(\varphi \wedge \psi) & = & LV(\varphi) \cup LV(\psi) \\
LV(\varphi \vee \psi) & = & LV(\varphi) \cup LV(\psi) \\
LV(\exists x.\varphi) & = & LV(\varphi) \setminus \{x\} \\
LV(TC(\lambda\, \vec{x}^{\vec{\sigma}}, \vec{y}^{\vec{\sigma}}.\varphi)(\vec{t}, \vec{u}) & = & (LV(\varphi) \setminus \{\vec{x}, \vec{y}\}) \cup LV(\vec{t}) \cup LV(\vec{u})
\end{array}$$

For example in the formula (skipping the annotations for brevity)
$$\exists x.x \leq (y \to z) \wedge t \leq x$$

the variable t is lonely, while x, y, z are not (x is bound and y, z occur inside a term).

Note that for balanced formulae, all free occurrences of a lonely variable are lonely.

4.1.4 Semantics

First we define a semantics for flat formulae. A Q-model is any poset R of which Q is a subposet. A valuation v assigns to each variable x an element $v(x) \in R$. Now we can define when a flat formula is satisfied in R by a valuation v ($R \models \varphi[v]$):

- $R \models (t_1 \leq_0 t_2)[v]$ iff $v(t_1) \leq_R v(t_2)$;
- $R \models (\varphi \wedge \psi)[v]$ iff $R \models \varphi[v]$ and $R \models \psi[v]$
- $R \models (\varphi \vee \psi)[v]$ iff $R \models \varphi[v]$ or $R \models \psi[v]$
- $R \models (\exists x^0.\varphi)[v]$ iff $R \models (\varphi[r/x])[v]$ for some $r \in R$;
- $R \models TC(\lambda\vec{x}, \vec{y}.\varphi)(\vec{t}, \vec{u})[v]$ iff there exists a positive integer k and vectors of elements of R $\vec{t} = \vec{r}_0, \vec{r}_1, \dots \vec{r}_k = \vec{u}$ such that $R \models (\varphi[\vec{r}_i/\vec{x}, \vec{r}_{i+1}/\vec{y}])[v]$ for $i = 0, 1, \dots, k-1$.

For closed formulae, we shall omit the valuation and write simply $R \models \varphi$.

Flat formulae can be used to express obstacles for retractibility. For example, any poset R extending the poset depicted in Fig. 4.1, retracts to it if and only if

$$R \not\models TC(\lambda x, y.((x \leq y \vee y \leq x) \wedge x \leq 1 \wedge x \leq 3 \wedge y \leq 1 \wedge y \leq 3))(0, 2).$$

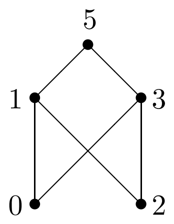


Figure 4.1: A poset which is TC-feasible but not Helly

4.1.1 Definition
Poset Q is called *TC-feasible* if there exists a flat TC-formula φ_Q such that for every R extending Q,

$$R \rhd Q \Leftrightarrow R \not\models \varphi_Q$$

Such φ_Q is called *a complete obstacle* for Q.

4.1.2 Theorem
Every absolute retract (and hence every Helly poset) is TC-feasible.

Proof: If Q is an absolute retract then a complete obstacle for it is a formula saing that some hole in Q is not separated, i.e. a disjunction over all holes[1] in Q of formulae stating that a particular hole is not separated. If

$$\mathcal{H} = \{(v_t, \delta_t^{+1}, \delta_t^{-1}) : t \in T\}$$

is a hole in Q, the fact that it is not separated in Q can be expressed as follows:

$$\exists x. \bigwedge\{d^{\varepsilon}(v_t, x) \leq \delta^{\varepsilon} \mid t \in T,\ \varepsilon = \pm 1\}$$

Distance judgements are also easy to express in our logic, e.g. we can rewrite

$$d^{+1}(p, q) \leq n$$

as

$$\exists x_1 \ldots \exists x_{n-1}. p \leq x_1 \wedge x_1 \geq x_2 \wedge \ldots \wedge x_{n-2} \leq x_{n-1} \wedge x_{n-1} \geq q$$

∎

From the above proof it follows that obstacles for absolute retracts can be expressed by existential formulae, i.e. without transitive closure operator. The poset depicted in Fig. 4.1 can also serve as an evidence that adding this operator really increases expressive power and that the class of TC-feasible posets is wider than Helly posets [PT96].

[1] Since Q is finite, the set of holes is also finite.

Before we present the semantics of arbitrary ATC-formulae, let us recall that variables are annotated with shapes which define how they should be valuated.

Let R be a poset, $v : X \to \mathcal{T}_R$. We say that v is compatible with φ if for every variable x free in φ, the shape of $v(x)$ corresponds to the annotation of x in φ. In what follows we shall consider only compatible valuations.

- $\mathcal{T}_R \models (t_1 \leq_\sigma t_2)[v]$ iff $v(t_1) \leq_{R_\sigma} v(t_2)$;
- $\mathcal{T}_R \models (\varphi \wedge \psi)[v]$ iff $\mathcal{T}_R \models \varphi[v]$ and $\mathcal{T}_R \models \psi[v]$
- $\mathcal{T}_R \models (\varphi \vee \psi)[v]$ iff $\mathcal{T}_R \models \varphi[v]$ or $\mathcal{T}_R \models \psi[v]$
- $\mathcal{T}_R \models (\exists x^\sigma.\varphi)[v]$ iff $\mathcal{T}_R \models (\varphi[r/x])[v]$ for some $r \in R_\sigma$;
- $\mathcal{T}_R \models TC(\lambda \vec{x}^{\vec{\sigma}}, \vec{y}^{\vec{\sigma}}.\varphi)(\vec{t}, \vec{u})[v]$ iff there exists a positive integer k and vectors of elements from $\mathcal{T}_R$ $\vec{t} = \vec{r}^0, \vec{r}^1, \ldots \vec{r}^k = \vec{u}$ such that $r_i^j \in R_{\sigma_i}$ for all relevant i, j and $\mathcal{T}_R \models (\varphi[\vec{r}^j/\vec{x}, \vec{r}^{j+1}/\vec{y}])[v]$ for $j = 0, 1, \ldots, k-1$.

4.1.3 Proposition
For flat φ we have

$$\mathcal{T}_R \models \varphi \textit{ iff } R \models \varphi$$

Proof: For atomic φ, by definition we have

$$\mathcal{T}_R \models t \leq_0 u \text{ iff } t \leq_R u \text{ iff } R \models t \leq u$$

Other cases follow by easy induction, basing on the fact that if $v : X \to \mathcal{T}_R$ is compatible with a flat formula φ then $v(x) \in R$ for every $x \in FV(\varphi)$ ∎

4.1.4 Definition
Let Q be a finite poset and Σ a system of inequalities over Q. We say that φ is a *semantical consequence* of Σ ($\Sigma \models \varphi$) if for every valuation v being a solution of Σ, $\mathcal{T}_Q \models \varphi[v]$.

4.1.5 Projections

First we define projections on shapes:

$$0 \downarrow i = 0, \qquad (\sigma_1 \to \sigma_2) \downarrow i = \sigma_i \quad i = 1, 2$$

Next we define projections on terms:

$$c \downarrow i = c \quad x^\sigma \downarrow i = x^{\sigma \downarrow i}, \; i = 1, 2$$

$$(t_1 \to t_2) \downarrow i = t_i, \; i = 1, 2$$

Now we define projections of ATC-formulae: $(\cdot) \downarrow 1, (\cdot) \downarrow 2 : ATC_Q \rightarrow ATC_Q$

$$\begin{array}{rcl}
(t \leq_0 u)\downarrow i & = & t \leq_0 u \\
(t \leq_{\sigma_1 \rightarrow \sigma_2} u) \downarrow 1 & = & (u \downarrow 1) \leq_{\sigma_1} (t \downarrow 1) \\
(t \leq_{\sigma_1 \rightarrow \sigma_2} u) \downarrow 2 & = & (t \downarrow 2) \leq_{\sigma_2} (u \downarrow 2) \\
(\varphi \wedge \psi)\downarrow i & = & (\varphi\downarrow i) \wedge (\psi\downarrow i) \\
(\varphi \vee \psi)\downarrow i & = & (\varphi\downarrow i) \vee (\psi\downarrow i) \\
(\exists x^{\sigma}.\varphi) \downarrow i & = & \begin{cases} \exists x^{\sigma\downarrow i}.\varphi \downarrow i & \text{if } x \in LV(\varphi) \\ \exists x^{\sigma}.\varphi \downarrow i & \text{otherwise} \end{cases} \\
(TC(\lambda\,\vec{x}^{\vec{\sigma}}, \vec{y}^{\vec{\sigma}}.\varphi)(t,u)) \downarrow i & = & TC(\lambda x^{\vec{\sigma}'}, y^{\vec{\sigma}'}.(\varphi \downarrow i))(t \downarrow i, u \downarrow i) \\
\text{where } \sigma_j' & = & \begin{cases} \sigma_j \downarrow i & \text{if } x_j \in LV(\varphi) \\ \sigma_j & \text{otherwise} \end{cases}
\end{array}$$

For $\pi = p_1p_2\ldots p_n \in \{1,2\}^*$ we shall write $\varphi \downarrow \pi$ for $(\ldots((\varphi \downarrow p_1) \downarrow p_2)\ldots) \downarrow p_n$.

4.1.5 Lemma
If φ is balanced, $LV(\varphi) = \emptyset$ and v is compatible with φ then $\mathcal{T}_R \models \varphi[v]$ iff $\mathcal{T}_R \models (\varphi \downarrow i)[v \downarrow i]$ for $i = 1, 2$.

Proof: by induction on φ. The case when $\varphi \equiv t \leq_\sigma u$ is trivial for $\sigma = 0$ and follows from definition of subtype order and projections for complex shapes (since $LV(\varphi) = \emptyset$, we have $t = t_1 \rightarrow t_2$ and $u = u_1 \rightarrow u_2$). The case when $\varphi \equiv \exists x^\sigma.\varphi_1$ is again trivial if $\sigma = 0$ and follows from the induction hypothesis if $x \notin LV(\varphi_1)$. Otherwise, if $\sigma = \sigma_1 \rightarrow \sigma_2$ then $\mathcal{T}_R \models \varphi[v]$ iff there exist $r_1 \in R_{\sigma_1}$, $r_2 \in R_{\sigma_2}$ such that $\mathcal{T}_R \models (\varphi_1[r_1 \rightarrow r_2/x])[v]$. Applying the induction hypothesis to the latter formula yields the thesis for this case. Conjunction and disjunction are obvious and TC can be handled similarly to $\exists$. ∎

4.1.6 Closures

Let $t \preceq u$ denote the formula $TC(\lambda x^\sigma, y^\sigma.x \leq y)(t,u)$, The closure of a formula φ (denoted $\overline{\varphi}$) is defined as follows:

$$\begin{array}{rcl}
\overline{t \leq u} & = & t \preceq u \\
\overline{\varphi \wedge \psi} & = & \overline{\varphi} \wedge \overline{\psi} \\
\overline{\varphi \vee \psi} & = & \overline{\varphi} \vee \overline{\psi} \\
\overline{\exists x^\sigma.\varphi} & = & \exists x^\sigma.\overline{\varphi} \\
\overline{TC(\lambda\,\vec{x}^{\vec{\sigma}}, \vec{y}^{\vec{\sigma}}.\varphi)(\vec{t},\vec{u})} & = & TC(\lambda x^{\vec{\sigma}}, y^{\vec{\sigma}}.(\overline{\varphi}))(t,u)
\end{array}$$

4.2 A proof system for ATC-formulae

As observed in [PT96], to decide satisfiability of a flat system Σ of inequalities over a poset Q for which we know a complete obstacle φ_Q, it is sufficient to check whether Σ satisfies an obstacle formula. This can be done in NLOGSPACE. Unfortunately, this result cannot be easily carried over to general systems, since there is no efficient (i.e. one that can be realized in polynomial time) method for checking satisfaction of a formula by a general system (as minimal solutions of such system can have exponential size).

In this section we introduce an inference system for ATC-formulae and prove its soundness. For the reasons outlined above, it is not complete. In fact it is designed so that for a fixed formula, we can check in polynomial time, whether it is derivable from a given system of inequalities. On the other hand we show the system is strong enough to be useful in deciding SSI for TC-feasible posets.

4.2.1 Inference rules

Let Σ be a weakly satisfiable system of inequalities over Q with all variables annotated according to the most general unifier of $\Sigma_\star$. Consider the inference system depicted in Fig. 4.2

4.2.1 Lemma
For every ATC-formula φ, if $\Sigma \vdash \varphi$ then $\Sigma \models \varphi$

Proof: By induction on the derivation of φ. The cases when the last rule was axiom, alternative or conjunction are obvious. If the last rule used was $(\downarrow)$, then the thesis follows directly from Lemma 4.1.5. ∎

4.2.2 Corollary
If φ is a complete obstacle for Q and $\Sigma \vdash \varphi$ then Σ is not satisfiable.

4.2.3 Theorem
For any fixed, flat ATC-formula φ, one can check in time polynomial in $|\Sigma|$, whether $\Sigma \vdash \varphi$.

Proof: First, observe that, if we erase annotations, then the only formulae which may occur in a derivation of $\Sigma \vdash \varphi$ are subformulae of φ with free variables instantiated by subterms of terms occurring in Σ. Hence, the number of such formulae is polynomial in $|\Sigma|$. On the other hand, the number of distinct shapes that may occur in such derivation is bounded by the size of Σ. Thus the number of formulae that may occur in the derivation is polynomial and we may check systematically for each of them (proceeding from

$$\Sigma \vdash t \leq u \text{ for } t \leq u \in \Sigma$$

$$\frac{\Sigma \vdash \varphi}{\Sigma \vdash \varphi \downarrow i}\ (\downarrow) \qquad LV(\varphi) = \emptyset$$

$$\frac{\Sigma \vdash \varphi \quad \Sigma \vdash \psi}{\Sigma \vdash \varphi \wedge \psi}\ (\wedge)$$

$$\frac{\Sigma \vdash \varphi_i}{\Sigma \vdash \varphi_1 \vee \varphi_2}\ (\vee)$$

$$\frac{\Sigma \vdash \varphi[t/x^\tau]}{\Sigma \vdash \exists x^\tau.\varphi}\ (\exists) \quad t : \tau$$

$$\frac{\Sigma \vdash \varphi[\vec{t}/\vec{x}^{\vec{\sigma}}, \vec{u}/\vec{y}^{\vec{\sigma}}]}{\Sigma \vdash TC(\lambda\, \vec{x}^{\vec{\sigma}}, \vec{y}^{\vec{\sigma}}.\varphi)(\vec{t}, \vec{u})}\ (TC_0)$$

$$\frac{\Sigma \vdash TC(\lambda\, \vec{x}^{\vec{\sigma}}, \vec{y}^{\vec{\sigma}}.\varphi)(\vec{t}, \vec{s}) \quad \Sigma \vdash TC(\lambda\, \vec{x}^{\vec{\sigma}}, \vec{y}^{\vec{\sigma}}.\varphi)(\vec{s}, \vec{u})}{\Sigma \vdash TC(\lambda\, \vec{x}^{\vec{\sigma}}, \vec{y}^{\vec{\sigma}}.\varphi)(\vec{t}, \vec{u})}\ (TC_S)$$

Figure 4.2: An inference system for ATC-formulae

bigger to smaller terms and from simpler to more complicated formulae), whether it is derivable from Σ. ∎

4.2.2 Simple formulae and restricted derivations

An annotated formula is called simple, if it contains no occurrences of TC or disjunction. Obviously no derivation of a simple formula can use rules for such constructs.

4.2.4 Lemma
If $\Sigma \vdash \exists x^{\sigma}.\varphi$, then there exist: a shape $\hat{\sigma}$, a term $t : \hat{\sigma}$, a formula $\hat{\varphi}$ and a path $\pi \in \{1,2\}^*$ such that

$$\begin{array}{rcl} \sigma & = & \hat{\sigma} \downarrow \pi \\ \varphi & = & \hat{\varphi} \downarrow \pi \\ \Sigma & \vdash & \hat{\varphi}[\hat{t}/x^{\hat{\sigma}}] \end{array}$$

Proof: This lemma is easily proved by induction on derivations. ∎

4.2.5 Lemma
The following rules are admissible:

$$\frac{\Sigma \vdash \exists y.\exists x.\varphi}{\Sigma \vdash \exists x.\exists y.\varphi}$$

$$\frac{\Sigma \vdash \exists \vec{z}.\exists y.\exists x.\varphi}{\Sigma \vdash \exists \vec{z}.\exists x.\exists y.\varphi}$$

$$\frac{\Sigma \vdash \varphi[t/z] \quad \Sigma \vdash \exists \vec{y}.\, \psi[t/z]}{\Sigma \vdash \exists z.\exists \vec{y}.\, (\varphi \wedge \psi)} \quad \vec{y} \notin FV(\varphi)$$

$$\frac{\Sigma \vdash \exists \vec{x}.\varphi[t/z] \quad \Sigma \vdash \exists \vec{y}.\, \psi[t/z]}{\Sigma \vdash \exists z.\exists \vec{y} \exists \vec{x}.(\varphi \wedge \psi)} \quad \begin{array}{l} \vec{y} \notin FV(\varphi) \\ \vec{x} \notin FV(\psi) \end{array}$$

Proof: This lemma can beproved by induction on derivations, using the previous lemma. ∎

The *simplification set* of formula φ is the set of simple formulae, defined as follows:

$$\begin{array}{rcl} [t \leq u] & = & \{t \leq u\} \\ [\varphi_1 \vee \varphi_2] & = & [\varphi_1] \cup [\varphi_2] \\ [\varphi_1 \wedge \varphi_2] & = & \{\psi_1 \wedge \psi_2 : \psi_1 \in [\varphi_1], \psi_2 \in [\varphi_2]\} \\ [\exists x.\varphi] & = & \{\exists x.\psi : \psi \in [\varphi]\} \end{array}$$

$$[TC(\lambda\vec{x}^{\vec{\sigma}},\vec{y}^{\vec{\sigma}}.\varphi)(\vec{t},\vec{u})] = \bigcup_{k\in\omega}\{\exists\vec{z}_1,\ldots,\vec{z}_k.\psi[\vec{t}/\vec{x},\vec{z}_1/\vec{y}]\wedge\psi[\vec{z}_1/\vec{x},\vec{z}_2/\vec{y}]\wedge \ldots\wedge\psi[\vec{z}_k/\vec{x},\vec{u}/\vec{y}] : \psi\in[\varphi]\}\}$$

4.2.6 Lemma
For every formula φ and $\psi \in [\varphi]$, we have (i) $FV(\psi) \subseteq FV(\varphi)$ and (ii) $LV(\psi) \subseteq LV(\varphi)$.

Proof: (i) follows by an easy induction over φ; (ii) follows from (i) and the fact that simplification does not change terms occuring in the formula (though some may be omitted, but it can only decrease LV). ∎

4.2.7 Lemma
For every ATC-formula φ, $\Sigma \vdash \varphi$ iff there exists a simple formula ψ belonging to the simplification set of φ such that $\Sigma \vdash \psi$.

Proof: This lemma is easily proved by structural induction over formulae, using Lemma 4.2.5. ∎

4.2.3 Canonical form of simple formulae

By canonical form of a simple formula we mean its prenex form. A formula in this form may be treated as a system of inequalities. Namely, for the formula $\varphi \equiv \exists\vec{x}.(t_1 \leq u_1 \wedge \ldots \wedge t_n \leq u_n)$, its corresponding system of inequalities is $\Delta(\varphi) = \{t_1 \leq u_1, \ldots, t_n \leq u_n\}$

If Γ and Δ are systems of inequalities, we say that $\Gamma \vdash \Delta$ if there is a formula φ such that $\Gamma \vdash \varphi$ and $\Delta = \Delta(\varphi)$.

4.2.4 Flat systems

Let Σ be a flat system of inequalities over Q. We shall write $Q \cup \Sigma$ as a shorthand for

$$\Sigma \cup \{t \leq u \mid Q \models t \leq u, \quad t, u \in Q\}.$$

Consider the set $Q_\Sigma = Q \cup var(\Sigma)$, preordered by the relation $\preceq$ defined as follows:

$$t \preceq u \text{ iff } Q \cup \Sigma \vdash t \preceq u$$

4.2.8 Lemma
Σ is satisfiable iff $\langle Q_\Sigma, \preceq\rangle$ retracts to $\langle Q, \leq\rangle$.

Proof: Let v be a solution of Σ. We will show that $v \cup id_Q$ is a retraction. The idempotence is obvious, so it only remains to prove monotonicity.

If $Q \cup \Sigma \vdash t \preceq u$ then there exist $t = r_0, r_1, \ldots, r_k = u \in var(\Sigma) \cup Q$ such that

$$Q \cup \Sigma \vdash r_i \leq r_{i+1} \quad i = 0, 1, \ldots, k-1$$

hence

$$Q \cup \Sigma \ni r_i \leq r_{i+1} \quad i = 0, 1, \ldots, k-1$$

Since v is a solution of Σ, we have

$$v(r_i) \leq_Q v(r_{i+1}) \quad i = 0, 1, \ldots, k-1$$

thus

$$v(t) \leq_Q v(u)$$

It is easy to see that any retraction $v : Q_\Sigma \to Q$ is a solution of Σ. ∎

For a given, finite Q we shall construct a formula $NGC(Q)$ such that $\Sigma \vdash NGC(Q)$ iff Σ is not ground consistent:

$$NGC(Q) \equiv \bigvee \{c \preceq d \mid Q \not\models c \leq d\}$$

4.2.9 Lemma
Let φ be a complete obstacle for Q. If $\Sigma \vdash NGC(Q)$ then Σ is not satisfiable. Otherwise Q_Σ retracts to Q iff $Q_\Sigma \not\models \varphi$.

Proof: From Lemma 4.2.1 it follows that if $\Sigma \vdash NGC(Q)$ then every solution of Σ must satisfy $NGC(Q)$; this is possible only if Σ has no solutions. If $\Sigma \not\vdash NGC(Q)$ then Q_Σ is an extension of Q and Q_Σ retracts to Q iff $Q_\Sigma \not\models \varphi$ since φ is a complete obstacle for Q. ∎

4.2.10 Lemma
For every ATC-formula ψ,

$$Q_\Sigma \models \psi \Leftrightarrow Q \cup \Sigma \vdash \bar{\psi}$$

(where $\bar{\psi}$ denotes the closure of ψ defined in section 4.1.6).

Proof: Note that, by definition of ordering on Q_Σ, for every $t, u \in Q_\Sigma$

$$Q_\Sigma \models t \leq u \Leftrightarrow Q \cup \Sigma \vdash t \preceq u$$

Now, the thesis follows by an easy induction on ψ. ∎

4.2.11 Lemma
Let φ be a complete obstacle for Q. For every flat system of inequalities Σ, Σ is satisfiable iff

$$Q \cup \Sigma \not\vdash \bar{\varphi} \vee NGC(Q)$$

Proof: This lemma is a simple consequence of previous three lemmas. ∎

4.2.5 Single-shaped systems and formulae

A system of inequalities is called *σ-shaped* if all its variables and inequalities are of the shape σ. Similarly, a formula is called *σ-shaped* if it is balanced and all its variables (free and bound) as well as all inequalities are annotated with σ. A system (formula) is called *single-shaped* if it is σ-shaped for some σ.

These notions will serve us as an intermediate level between flat and general systems of inequalities and will facilitate the proof of the main theorem.

4.2.12 Definition
Let $\sigma = \sigma_1 \to \sigma_2$ and let $\Sigma = \{t_1 \leq u_1, \ldots, t_n \leq u_n\}$ be a σ-shaped system of inequalities. For $i = 1, 2$, we define

$$\Sigma \Downarrow i = \{(t_1 \leq u_1) \downarrow i, \ldots, (t_n \leq u_n) \downarrow i\}$$

4.2.13 Lemma
Let φ be a complete obstacle for Q and Σ be single-shaped. Σ is satisfiable iff

$$Q \cup \Sigma \not\vdash \varphi \vee NGC(Q)$$

Proof: Before we delve into technicalities, let us explain the intuition behind this lemma. A σ-shaped system of inequalities over Q may be viewed as a flat system over Q_σ, which is again a TC-feasible poset. Even though φ is not a complete obstacle for Q_σ, one can easily construct a formula φ^σ which is such an obstacle, and having the property that $\Sigma \vdash \varphi$ iff $\Sigma \vdash \varphi^\sigma$. But as this line of proof is technically more complicated and needs several technical lemmas similar to 4.2.5, we shall prove this lemma in a slightly different way.

If $\Sigma \vdash \varphi$ then obviously Σ is not satisfiable (see Corollary 4.2.2). Thus it remains to prove that if Σ is not satisfiable then $\Sigma \vdash \varphi$. Let Σ be σ-shaped. We shall proceed by induction on σ. If Σ is flat then the thesis follows from the Lemma 4.2.11. On the other hand, if $\sigma = \sigma_1 \to \sigma_2$, Σ is unsatisfiable iff either $\Sigma \Downarrow 1$ or $\Sigma \Downarrow 2$ is.

Let us assume that $Q \not\models \Sigma \Downarrow 2$ (the other case is handled dually). By completeness of φ, we have that $\Sigma \Downarrow 2 \vdash \varphi$. We shall show that this implies $\Sigma \vdash \varphi$. First we shall prove that for every formula ψ derivable from $\Sigma \Downarrow 2$ without using $(\downarrow)$, there exists a formula $\psi \uparrow 2$ such that

1. $\Sigma \vdash \psi \uparrow 2$
2. $(\psi \uparrow 2) \downarrow 2 = \psi$
3. $LV(\psi \uparrow 2) = \emptyset$ iff $LV(\psi) = \emptyset$

- If $\psi \equiv t \leq u$ then $t \leq u \in \Sigma \Downarrow 2$, since the derivation does not contain $(\downarrow)$. Hence there is $t' \leq u'$ in Σ such that $(t' \leq u') \downarrow 2 = t \leq u$.

- If $\psi \equiv \exists x^\tau.\psi'$ and $\Sigma \Downarrow 2 \vdash \psi'[t/x]$ then $\Sigma \vdash (\psi'[t/x]) \uparrow 2$. Let ψ'' be such that
$$\psi''[t/x] = (\psi'[t/x]) \uparrow 2$$
and let
$$\psi \uparrow 2 = \begin{cases} \exists x^{\sigma_1 \to \sigma_2}.(\psi'') & \text{if } \tau = \sigma_2 \\ \exists x^\tau.\psi'' & \text{otherwise} \end{cases}$$
The correctness of the above definition follows from the fact that, since $\Sigma \Downarrow 2$ is σ_2-shaped, $x \in LV(\psi')$ iff $\tau = \sigma_2$.

- The TC case is handled very similarly (cf. the definition of projection in section 4.1.5.

- The cases of conjunction and disjunction are trivial.

If the derivation of the obstacle φ from $\Sigma \Downarrow 2$ does not contain $(\downarrow)$, the thesis follows immediately. Otherwise consider a subderivation ending with an application of the rule $(\downarrow)$

$$\dfrac{\dfrac{\vdots}{\Sigma \Downarrow 2 \vdash \psi}}{\Sigma \Downarrow 2 \vdash \psi \downarrow i}$$

and such that it contains no other application of this rule. We have

1. $\Sigma \vdash \psi \uparrow 2$
2. $(\psi \uparrow 2) \downarrow 2 = \psi$
3. $LV(\psi) = \emptyset$
4. $LV(\psi \uparrow 2) = \emptyset$

Hence

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \Sigma \vdash \psi \uparrow 2\end{array}}{\Sigma \vdash \psi}}{\Sigma \vdash \psi \downarrow i}$$

Thus we have proved the desired thesis. ∎

4.2.14 Definition
A system $\hat{\Sigma}$ is called a σ-view of Σ if the following conditions hold:

1. $\Sigma \vdash \hat{\Sigma}$
2. $\hat{\Sigma}$ is σ-shaped
3. For every σ-shaped Δ, if $\Sigma \vdash \Delta$ then $\hat{\Sigma} \vdash \Delta$.

4.2.15 Lemma
For every system Σ and shape σ minimal in Σ there exists a σ-view of Σ.

Proof: Let σ be a shape minimal in Σ, ρ be a shape of some inequality in Σ and π be a path such that $\rho \downarrow \pi = \sigma$. Further, let Σ_ρ denote the set of ρ-shaped inequalities in Σ and ψ_ρ^Σ be the formula

$$\psi_\rho^\Sigma = \exists \vec{x}. \bigwedge_{t \le u \in \Sigma_\rho} t \le u$$

where the quantification is over all variables occurring in Σ_ρ which have shape different from σ. Obviously, the formula $(\psi_\rho^\Sigma \downarrow \pi)$ is σ-shaped, and because of minimality of σ it is derivable from Σ.

Now it is easily seen that

$$\Delta_\sigma^\Sigma = \Delta(\bigwedge\{(\psi_\rho^\Sigma \downarrow \pi) \mid \rho \downarrow \pi = \sigma\})$$

is a σ-view of Σ. ∎

4.2.6 General systems

4.2.16 Theorem
Let φ be the complete obstacle for Q. For every system of inequalities Σ, Σ is satisfiable iff it is weakly satisfiable and

$$Q \cup \Sigma \not\vdash \varphi \vee NGC(Q)$$

Proof: The ($\Rightarrow$) implication is obvious. The opposite implication is proved by induction on the number of equivalence classes of $\sim$ defined on $var(\Sigma)$ as follows

$$x \sim y \quad \text{iff} \quad \Sigma_* \models x = y$$

Suppose first that the quotient set $var(\Sigma)/_\sim$ has only one element σ. Then every inequality in Σ is either σ-shaped, or of the form $t_1 \to t_2 \le u_1 \to u_2$, or $p \le q$ with $p, q \in Q$. Thus it is easy to construct (by decomposition of complex inequalities and removing inequalites between constants from Q) a system Σ' which is σ-shaped and equivalent (in the sense of satisfiability as well as derivability of flat formulae) to Σ. Satisfiability of Σ' (and hence Σ) follows from Lemma 4.2.13.

Now let us assume that $var(\Sigma)/_\sim$ has $n+1$ elements and let $y \in var(\Sigma)$ be such that for no $z \in var(\Sigma)$ there is a term τ with $|\tau| > 1$, $z \in var(\tau)$ and $\Sigma_* \models \tau = y$. If there is no such y, then one easily finds a term τ such that $|\tau| > 1$, $z \in var(\tau)$ and $\Sigma_* \models \tau = z$. This would contradict weak satisfiability of Σ.

Let

$$[y] = \{z \in var(\Sigma) | z \sim y\} = \{y_1, \ldots, y_k\}$$

Let σ be the shape assigned to y by $mgu(\Sigma_\star)$, and let $\hat{\Sigma}$ be a σ-view of Σ.

Again, the satisfiability of $\hat{\Sigma}$ follows from the Lemma 4.2.13. Let $\hat{v} : [y] \to \mathcal{T}_\sigma$ be a solution of $\hat{\Sigma}$ and let

$$\Sigma_1 = \hat{v}(\Sigma) = \{\hat{v}(\tau) \leq \hat{v}(\rho) : \tau \leq \rho \in \Sigma\}$$

In the above definition $\hat{v}$ acts as identity on variables other than those in $[y]$. Let $\sim_1$ be the equivalence relation associated with Σ_1. One can prove that

$$|var(\Sigma_1)/_{\sim_1}| = n$$

$$\Sigma_1 \text{ is weakly satisfiable}$$

To complete the proof we need to prove that

$$\Sigma_1 \not\vdash \varphi \vee NGC(Q)$$

To do this we shall prove that for every flat Δ derivable from Σ_1, $Q \models \Delta$. On the other hand we have that

$$Q \not\models \varphi \vee NGC(Q)$$

For every flat Δ derivable from Σ_1 there exists a σ-shaped Δ' derivable from Σ such that Δ is derivable from $\hat{v}(\Delta')$. Since $\hat{\Sigma}$ is a σ-view of Σ, we also have $\hat{\Sigma} \vdash \Delta'$. Thus $\hat{v}(\hat{\Sigma}) \vdash \hat{v}(\Delta')$. But since $\hat{v}$ is a solution of $\hat{\Sigma}$, we have $Q \models \Delta'$, which we wanted to prove. ∎

4.2.17 Corollary
For any TC-feasible Q and Σ — a system of inequalities over Q one can check in time polynomial in $|\Sigma|$, whether Σ is satisfiable.

Proof: By Theorem 4.2.16 there is a flat ATC formula φ_Q depending only on Q and such that Σ is satisfiable iff φ is not derivable from Q, which by Theorem 4.2.3 can be checked in polynomial time. ∎

Chapter 5

Subtyping and Alternation

Contents

5.1 Introduction

The aim of this chapter is to establish further links between SSI and FLAT-SSI,providing some evidence in favour of the following conjecture:

5.1.1 Conjecture
Given a poset Q such that Q-FLAT-SSI is complete for $NTM(s,t)$, Q-SSI is complete for $ATM(s,t)$.

In our opinion, the ‘nondeterminism vs alternation’ concept constitutes a framework within which various complexity phenomena bound with subtyping can be explained. Sure enough, there is still a lot of open questions and gaps to be filled, but we present it with hope that it will encourage further research in this area. One example would be the apparent ’gap’ in the poset hierarchy. So far we know no posets for which SSI is NP-complete or FLAT-SSI — P-complete. Within our framework, the explanation for this gap is provided by the fact that (unless P=NP or NP=PSPACE) NP is not an alternating complexity class and (unless P=NL or P=NP), P is not a nondeterministic complexity class.

5.2 Motivating examples

First let us look at several examples known so far that supporting the thesis that arrows in the systems of inequalities correspond on the complexity level exactly to the transition from nondeterministic classes to corresponding alternating classes. This is at the same time a resume of current knowledge about the complexity of SSI:

1. If Q is discrete, then
   - Q-FLAT-SSI is in NLOGSPACE[1];
   - Q-SSI is equivalent to the unification, and hence AL-complete.
2. If Q is a disjoint union of lattices (but not discrete), then
   - Q-FLAT-SSI is NLOGSPACE-complete [Ben94];
   - Q-SSI is ALOGSPACE-complete [Tiu92].
3. If Q is a non-discrete Helly poset, then
   - Q-FLAT-SSI is NLOGSPACE-complete [Ben93, Ben94];

[1]the problem whether it is NLOGSPACE-hard is equivalent to a known open problem in complexity, whether SYMLOGSPACE=NLOGSPACE

- Q-SSI is ALOGSPACE-complete [Ben93].

4. If Q is a non-discrete TC-feasible poset, then
   - Q-FLAT-SSI is NLOGSPACE-complete [PT96];
   - Q-SSI is ALOGSPACE-complete (Corollary 4.2.17).

5. If Q is an n-crown ($n > 1$), then
   - Q-FLAT-SSI is NP-complete [PT96];
   - Q-SSI is AP-complete [Tiu92, Fre97].

5.3 Encoding alternation

In this section we show that the result of [Tiu92] (AP-hardness of SSI for crowns) can be generalized stating that for all posets for which FLAT-SSI is NP-hard, SSI is AP-hard. To this end, we construct an encoding for QBF as an SSI, given encoding of SAT as FLAT-SSI.

First let us make some assumptions about encodings of instances of SAT as systems of inequalities. Later we show how these assumptions can be either removed or replaced. Intuitively, these assumptions express the requirement that whenever there exists a simulation of NTM, there exists one which is "regular" enough to be transformed to a simulation of an ATM. This intuition is formalized in the following

5.3.1 Definition
Let $\varphi = \varphi(\vec{x})$ be a 3-CNF[2] propositional formula with variables $\vec{x} = x_1, \dots x_n$ (and no other)

We say that a flat system of inequalities Σ_φ encodes φ if there exist variables $z_1, \dots, z_n$ and constants $\vec{c}$ such that for every $p_1, \dots, p_n \in \{0, 1\}$

$$\models \varphi[\vec{p}/\vec{x}] \iff \Sigma_\varphi[\vec{c}/\vec{z}] \text{ is satisfiable}$$

We say the encoding is *symmetric*, if there exists an antimonotonic bijection $f : Q \to Q$ that extends to an antimonotonic bijection of (the poset corresponding to) Σ_φ onto itself and such that $c_i^1 = f(c_i^0)$ for $i = 1, \dots, n$.

5.3.2 Theorem
Let Q be a poset such that Q-FLAT-SSI is complete for NP under symmetric *reductions. Then Q-SSI is complete for AP.*

[2]3-CNF means a conjunctive normal form with 3 disjuncts in each clause.

Proof: Since [Fre97] presents an AP-algorithm for deciding SSI for an arbitrary finite poset, we need to prove hardness only. Let

$$\forall x_n \exists y_n \ldots \forall x_1 \exists y_1 \ \varphi$$

be an instance of QBF, φ contains no quantifiers and is in 3-CNF.

Let Σ_φ be a symmetric encoding of φ. We show how to construct a system of inequalities Σ_k such that

$$\psi_k \text{ holds} \iff \Sigma_k \text{ is satisfiable}$$

where

$$\psi_k = \exists x_n \exists y_n \ldots \exists x_{k+1} \exists y_{k+1} \forall x_k \exists y_k \ldots \forall x_1 \exists y_1 \ \varphi$$

The construction of Σ_k is by induction on k, the number of quantifier alternations in ψ_k.

Let q be the smallest positive integer such that $f^q = id$ (such q must exist since Σ is finite, moreover it cannot be greater than $|\Sigma|$).

In what follows we use a with sub- or super-scripts. These are new variables. We will also use new variables $[u]^{i,j}_k$, where $0 \le k \le n$, $i, j \in Q$ and u is a propositional variable of φ. The variable $[u]^{i,j}_k$ is a version of $[u]^{i,j}$, lifted to level k. The variable a^i_k, which we use below, represents constant i lifted to level k.

Let us first define sets Δ_k, for $0 \le k \le n$.

$$\Delta_0 = \{\, a^{i,j}_{0,0} = a^j_0 \mid i, j \in P \,\} \cup \{\, a^i_0 = i \mid i \in P \,\}$$

For $k < n$, Δ_{k+1} is Δ_k plus the equations (5.1–5.4) below, with i, j ranging over Q.

$$a^i_{k+1} = a^{f(i)}_k \to a^i_k \tag{5.1}$$

For $k + 1 < p \le n$ and $z_p \in \{x_p, y_p\}$,

$$f^i(z_{p,k+1}) = f^{i+1}(z_{p,k}) \to f^i(z_{p,k}) \quad \text{for } i = 0, \ldots q-1 \tag{5.2}$$

For $1 \le p \le k$,

$$a^{i,j}_{p,k+1} = a^{f(j),f(i)}_{p,k} \to a^{i,j}_{p,k} \tag{5.3}$$

$$a^{i,j}_{k+1,k+1} = a^j_{k+1} \tag{5.4}$$

For every $k \ge 0$, let $\hat{\Sigma}_k$ be the system of inequalities obtained from $\hat{\Sigma}$ by replacing every variable $[u]^{i,j}$ of $\hat{\Sigma}$ by $[u]^{i,j}_k$, and replacing the constant $i \in Q$ by a (new) variable a^i_k. Hence, there are no constants in $\hat{\Sigma}_k$.

Finally we set $\Sigma_{k+1} = \Delta_{k+1} \cup \hat{\Sigma}_{k+1}$ plus the equation (5.5) with i, j ranging over Q and $1 \leq p \leq k+1$.

$$z_{p,k+1} = a_{p,k+1}^{c_p^0, c_p^1} \tag{5.5}$$

The thesis follows from the following lemmas:

5.3.3 Lemma
Let $V_k = \{x_{k+1}, y_{k+1}, \ldots, x_n, y_n\}$. For all $k \geq 0$, and for every function $\xi : V_k \to \{0,1\}$, $\Sigma_{k+1} \cup \{ z_k = a_k^{c_k^{\xi(v)}} \mid v \in V_{k+1} \}$ is satisfiable iff for every $i \in \{0,1\}$, $\Sigma_k \cup \{ z_k = a_k^{c_k^{\xi(v)}} \mid v \in V_k \} \cup \{z_{k+1,k} = a_k^{c_i}\}$ is satisfiable.

Proof: Take Σ_{k+1}. Let u be one of $z_1, \ldots, z_n$. The inequalities in $\hat{\Sigma}_{k+1}$ compare u_{k+1} with some a_{k+1}^l, hence by (5.1), the former has to be expanded introducing two new variables. We use a special naming convention for the new variables introduced by this expansion, so that it will be easier to follow the proof. Let us choose the substitution

$$u_{k+1} = f(u_k^0) \to u_k^1 \tag{5.6}$$

First, we shall show that $\hat{\Sigma}_{k+1}$ is equivalent to two copies of $\hat{\Sigma}_k$, one for u^0's and the other for u^1's. Indeed, $\hat{\Sigma}_{k+1}$ is equivalent to

$$f(\hat{\Sigma}_k[\vec{z}_k^0/\vec{z}_k]) \cup \hat{\Sigma}_k[\vec{z}_k^1/\vec{z}_k]$$

For $k+1 < p \leq n$, by (5.2) and (5.6) we get:

$$x_{p,k}^1 = x_{p,k} \tag{5.7}$$

and

$$f(x_{p,k}^0) = f(x_{p,k}) \tag{5.8}$$

Since f is a bijection, it follows that the variables x_p^0 and x_p^1 are equated for $k+1 < p \leq n$ and we can assume that we are dealing just with one copy x_p. A similar statement holds for $y_{k+2}, \ldots, y_n$.

By (5.5) (for $p = k+1$), (5.6), (5.4) and (5.1) we obtain

$$f^i(x_{k+1,k}^1) = f^i(a_k^{c_p^1}) \tag{5.9}$$

and

$$f^{i+1}(x_{k+1,k}^0) = f^i(a_k^{f(c_p^0)}) \tag{5.10}$$

Substituting $i = 0,\ j = 1$ in (5.9) yields $x_{k,k+1]}^1 {=} a_k^{c_{k+1}^1}$. Similarly, substituting $i = 0,\ j = q-1$ in (5.10) we get $x_{k+1,k}^0 = a^{c_p^0}$. Putting these two together we obtain for $i = 0, 1$,

$$x_{k,k+1}^i = a_k^{c_{k+1}^i} \tag{5.11}$$

By (5.5), (5.3) and (5.6) we can conclude that for $l = 0, 1$, $1 \leq p \leq k$ and $0 < i \leq q$,

$$f^i(x^l_{p,k}) = f^i(a_k^{c^1_p})$$

Thus we have shown that Σ_{k+1} is equivalent to (5.11) plus two copies of Σ_k, one copy in which for every $1 \leq p \leq k$, every x_p and every y_p has been replaced by x^0_p and y^0_p, respectively; and the other in which x_p and every y_p has been replaced by $(x_p)_1$ and $(y_p)_1$. This completes the proof of the lemma. ∎

For $0 \leq k \leq n$ let

$$\varphi_k = \forall x_k \exists y_k \ldots \forall x_1 \exists y_1 \ \varphi$$

Hence, free variables of φ_k are among $V_k = \{x_{k+1}, y_{k+1}, \ldots, x_n, y_n\}$. The following result shows correctness of the choice of Σ_k.

5.3.4 Lemma

For every $0 \leq k \leq n$ and for every valuation $\xi : V_k \to \{0, 1\}$, ξ satisfies φ_k iff $\Sigma_k \cup \{z_j = a_k^{c_j^{\xi(z_j)}} \mid z_j \in V_k\}$ is satisfiable.

Proof: The proof is by induction on k. For $k = 0$, it is enough to observe that φ_0 is φ and the statement follows from the proof of NP-hardness of the flat case.

Now, in order to complete the proof let us take any truth assignment $\xi : V_{k+1} \to \{0, 1\}$, and for $i, j \in \{0, 1\}$ let $\xi_{i,j} : V_k \to \{0, 1\}$ be an extension of ξ such that $\xi_{i,j}(x_{k+1}) = i$ and $\xi_{i,j}(y_{k+1}) = j$. Then we have

$$\xi \text{ satisfies } \varphi_{k+1} \tag{5.12}$$

iff

$$\forall i \in \{0, 1\} \exists j \in \{0, 1\} \xi_{i,j} \text{ satisfies } \varphi_k \tag{5.13}$$

iff $\forall i \in \{0, 1\} \exists j \in \{0, 1\}$

$$\begin{aligned} \Sigma_k \quad & \cup \quad \{ v_k = a_k^{c_k^{\xi(v)}} \mid v \in V_k \} \\ & \cup \quad \{x_{k+1,k} = a_k^{c^i_{k+1}}, \ y_{k+1,k} = a_k^{c^j_{k+1}}\} \\ & \qquad \text{is satisfiable} \end{aligned} \tag{5.14}$$

iff $\forall i \in \{0, 1\}$

$$\begin{aligned} \Sigma_k \quad & \cup \quad \{ v_k = a_k^{c_k^{\xi(v)}} \mid v \in V_k \} \\ & \cup \quad \{x_{k+1,k} = a_k^{c^i_{k+1}} \} \\ & \qquad \text{is satisfiable} \end{aligned} \tag{5.15}$$

iff

$$\Sigma_{k+1} \cup \{\, v_k = a_k^{c_k^{\xi(v)}} \mid v \in V_{k+1}\} \text{ is satisfiable} \tag{5.16}$$

Equivalence of (5.13) and (5.14) follows from the induction assumption. Equivalence of (5.15) and (5.16) follows from Proposition 5.3.3. ∎

Part II

Typability in ML with subtyping

Chapter 6

Subtyping in ML

Contents

6.1 Introduction

In this part we discuss problems concerning combining subtyping with the ML (also known as Hindley-Milner) type discipline. This discipline, using shallow polymorphic (also called “rank 1”) type schemes[1], constitutes the kernel of type systems of almost all current functional languages, including Standard ML, CAML, Haskell, Clean and Miranda. Exceptions are of course languages that are not statically typed (e.g. Lisp family) or those with restricted use of higher order functions (like Erlang). There are two ways of extending a polymorphic type discipline with subtyping: the simpler one modifies only the typing rules while preserving the syntax of types, the other introduces subtyping constraints into type syntax (thus making instantiation of bound type variables subject to satisfaction of these constraints). In this chapter, we concentrate on the former, presenting an algebraic characterisation of typability in this system.

6.2 Preliminaries

6.2.1 Terms

We concentrate on a subset of ML terms essential for type analysis:

$$M ::= c \mid x \mid \lambda x.M \mid M_1 M_2 \mid \textbf{let}\ x = M_1\ \textbf{in}\ M_2$$

(x stands for variables and c for constants)

6.2.2 Types and type schemes

Given a set of type variables ($\alpha, \beta, \gamma, \ldots$) and a (finite) set of type constants (like *char,int,real,...*), we define the set of (monomorphic) types

$$\tau ::= \kappa \mid \alpha \mid \tau \to \tau$$

where κ stands for type constants.

Further, we define the set of (polymorphic) type schemes

$$\sigma ::= \forall \alpha_1 \ldots \alpha_n . \tau$$

In the sequel we shall use the abbreviation $\forall \vec{\alpha}.\tau$, and a notational convention that σ (possibly with indices) will stand for type schemes and τ, ρ (possibly with indices) for (monomorphic) types.

If $\alpha \notin FV(\sigma)$ then $\forall \alpha.\sigma$ is called an empty binding. A type scheme containing only empty bindings is called singular.

[1]that is with quantification at top level only

6.2.3 Subtype partial orders

We assume there is some predefined partial ordering $\leq_\kappa$ on the type constants.

We introduce a system of subtyping for ML types being a restriction of the Mitchell's system [Mit88] to ML types, enriched with the subtyping between constants. The system derives formulas of the form $\sigma \leq \tau$, where σ and τ are type schemes.

Axioms:

(refl) $\sigma \leq \sigma$

(inst) $\forall\vec{\alpha}.\,\sigma \leq \forall\vec{\beta}.\,\sigma[\vec{\rho}/\vec{\alpha}]$, ρ_i are types; $\beta_i \notin FV(\forall\vec{\alpha}.\,\sigma)$

Rules:

$$(\text{const})\quad \frac{\kappa_1 \leq_\kappa \kappa_2}{\kappa_1 \leq \kappa_2}$$

$$(\rightarrow)\quad \frac{\rho' \leq \rho \quad \tau \leq \tau'}{\rho \to \tau \leq \rho' \to \tau'} \qquad\qquad (\forall)\quad \frac{\sigma \leq \sigma'}{\forall\alpha.\,\sigma \leq \forall\alpha.\,\sigma'}$$

$$(\text{trans})\quad \frac{\sigma \leq \sigma_1 \quad \sigma_1 \leq \sigma'}{\sigma \leq \sigma'}$$

We write $\vdash \sigma \leq \tau$ to indicate that $\sigma \leq \tau$ is derivable in the above system. We shall also write $\vdash_\sigma$ for derivability without using the axiom **(inst)** (and use $\vdash_\sigma \sigma_1 \leq \sigma_2$ as a a shorthand for $\sigma_1 \leq_\sigma \sigma_2$) and $\vdash_\tau$ for derivability without mentioning type schemes at all.

We shall use the symbol $\preceq$ to denote the relation generated by the **(inst)** axiom.

It is worthwhile to observe that this is not a conservative restriction of Mitchell's system, i.e. if we allow substituting polymorphic types in **(inst)**, we can infer more inequalities between ML types. A simple example here is the inequality

$$\forall\alpha.((\alpha \to \beta) \to \alpha) \leq (\alpha \to \beta) \to \gamma$$

which is derivable in the Mitchell's system but (as follows from Theorem 6.2.7) not in ours. On the other hand an important consequence of allowing only monomorphic instance is the following

6.2.1 Proposition ([OL96])
The relation $\leq$ is decidable.

In fact one can even prove that it is decidable in polynopmial time.

6.2.2 Lemma
If $\vdash \sigma \leq \sigma'$ and σ is singular then

1. *$FV(\sigma) = FV(\sigma')$*
2. *the derivation contains only singular type schemes (in particular $\sigma'$is singular).*

Proof: By induction on the derivation. The case of **(refl)** is trivial. If the derivation consists of an instance of the axiom **(inst)** then it looks like

$$\forall\vec{\alpha}.\,\sigma \leq \forall\vec{\beta}.\,\sigma[\vec{\rho}/\vec{\alpha}] \quad \beta_i \notin FV(\forall\vec{\alpha}.\,\sigma)$$

Since $\alpha \notin FV(\sigma)$ then $\sigma[\vec{\rho}/\vec{\alpha}] = \sigma$ and $\beta_i \notin FV(\sigma)$. Hence $FV(\forall\vec{\alpha}.\,\sigma) = FV(\forall\vec{\beta}.\,\sigma)$ and $\forall\vec{\beta}.\,\sigma$ is quasi-mono.

The cases when the derivation ends with $(\rightarrow)$ or (const) are obvious. Let us then consider a proof ending with an application of the rule $(\forall)$:

$$\frac{\sigma \leq \sigma'}{\forall\alpha.\,\sigma \leq \forall\alpha.\,\sigma'}$$

If $\forall\vec{\alpha}.\,\sigma$ is singular then so is σ. By induction assumption $FV(\sigma) = FV(\sigma')$ and the derivation of $\sigma \leq \sigma'$ contains only singular type schemes. Thus $\alpha \notin FV(\sigma')$, $FV(\forall\alpha.\,\sigma) = FV(\forall\alpha.\,\sigma')$ and whole derivation contains only singular type schemes.

Finally, consider the case when the last rule used was (trans):

$$\frac{\sigma \leq \sigma_1 \quad \sigma_1 \leq \sigma'}{\sigma \leq \sigma'}$$

By the induction assumption $FV(\sigma) = FV(\sigma_1)$ and σ_1 is quasi-mono. Thus we can reapply the induction assumption to the inequality $\sigma_1 \leq \sigma'$, concluding that $FV(\sigma') = FV(\sigma_1)$ and the derivation contains only singular type schemes. From that the thesis follows. ∎

6.2.3 Lemma
If $\vdash \tau \leq \tau'$ (with τ, τ' monomorphic) then there is a derivation of this inequality which contains only monomorphic types.

Proof: By the Lemma 6.2.2, the derivation of $\vdash \tau \leq \tau'$ may contain only quasi-monotypes. We shall prove that erasing all empty bindings in such a derivation yields a valid derivation (in this proof $\mathit{erase}(\sigma)$ will mean σ with empty bindings erased). As usual, we shall proceed by induction on the derivation. Here the only interesting cases to be considered are **(inst)** and $(\forall)$. Consider an instance of the former:

$$\forall\vec{\alpha}.\,\sigma \leq \forall\vec{\beta}.\,\sigma[\vec{\rho}/\vec{\alpha}] \quad \beta_i \notin FV(\forall\vec{\alpha}.\,\sigma)$$

Since $\alpha_i \notin FV(\sigma)$ for all i, we have $\beta_i \notin FV(\sigma)$ and erasing empty bindings yields an instance of the axiom **(refl)**.

Now consider an instance of the rule $(\forall)$:

$$\frac{\sigma \leq \sigma'}{\forall\alpha.\,\sigma \leq \forall\alpha.\,\sigma'}$$

Since both quantifiers are empty, after erasure this rule becomes the identity step:

$$\frac{\mathit{erase}(\sigma) \leq \mathit{erase}(\sigma')}{\mathit{erase}(\sigma) \leq \mathit{erase}(\sigma')}$$

∎

6.2.4 Normalization

6.2.4 Lemma
The relation $\preceq$ is reflexive and transitive.

6.2.5 Lemma
If $\sigma \leq_\sigma \sigma_1 \preceq \sigma'$ then there exists σ_2 such that

$$\sigma \preceq \sigma_2 \leq_\sigma \sigma'.$$

Proof: If $\sigma \leq_\sigma \sigma_1$ then there exist monomorphic types τ, τ_1 such that

$$\sigma \equiv \forall\vec{\alpha}.\tau \tag{6.1}$$
$$\sigma_1 \equiv \forall\vec{\alpha}.\tau_1 \tag{6.2}$$
$$\tau \leq_\sigma \tau_1 \tag{6.3}$$

Thus $\sigma' \equiv \forall\vec{\beta}.\,\tau[\vec{\rho}/\vec{\alpha}]$, $\beta_i \notin FV(\sigma_1)$. It is easy to see that $\sigma_1 \equiv \forall\vec{\beta}.\,\tau[\vec{\rho}/\vec{\alpha}]$ fulfills conditions of the thesis. ∎

6.2.6 Lemma
If $\sigma_0 \preceq \sigma_1$ then $\forall\gamma.\,\sigma_0 \preceq \forall\gamma.\,\sigma_1$.

Proof: By the definition of $\preceq$, we have $\sigma_0 \equiv \forall\vec{\alpha}.\,\sigma$ and $\sigma_1 \equiv \sigma[\vec{\rho}/\vec{\alpha}]$ for some σ, ρ. We have to consider two cases:

1. $\gamma \in FV(\forall\vec{\alpha}.\,\sigma)$. Then we have

$$\forall\gamma.\forall\vec{\alpha}.\,\sigma \preceq \forall\vec{\alpha}.\,\forall\gamma.\,\sigma[\gamma/\gamma, \vec{\alpha}/\vec{\alpha}] \equiv \forall\vec{\alpha}.\,\forall\gamma.\,\sigma \preceq \forall\gamma.\sigma[\vec{\rho}/\vec{\alpha}]$$

2. $\gamma \notin FV(\forall\vec{\alpha}.\,\sigma)$. Then we have

$$\forall\gamma.\forall\vec{\alpha}.\,\sigma \preceq \forall\vec{\alpha}.\,\sigma[\gamma/\gamma] \equiv \forall\vec{\alpha}.\,\sigma \preceq \forall\gamma.\,\sigma[\vec{\rho}/\vec{\alpha}]$$

In both cases the thesis follows from the transitivity of $\preceq$. ∎

6.2.7 Theorem (Normalization for $\leq$)
If $\vdash \sigma \leq \sigma'$ then there exists σ_1 such that

$$\sigma \preceq \sigma_1 \leq_\sigma \sigma'$$

Proof: Again we proceed by induction on the derivation. The basic cases i.e. axioms, and (const) are trivial. The rule (trans) can be handled by Lemma 6.2.5.

If the last rule in the derivation was ($\rightarrow$) then all components must be monomorphic, and by Lemma 6.2.3 there is a derivation of the inequality in $\vdash_\sigma$.

Having said that, we only have to consider the case when the last rule was ($\forall$):

$$\frac{\sigma \leq \sigma'}{\forall\alpha.\,\sigma \leq \forall\alpha.\,\sigma'}$$

By the induction assumption, there exists σ_1 such that

$$\sigma \preceq \sigma_1 \leq_\sigma \sigma'.$$

But then, by Lemma 6.2.6 we have that

$$\forall\vec{\alpha}.\,\sigma \preceq \forall\vec{\alpha}.\,\sigma_1$$

On the other hand, obviously if $\sigma_1 \leq_\sigma \sigma'$ then $\forall\vec{\alpha}.\,\sigma_1 \leq_\sigma \forall\vec{\alpha}.\,\sigma'$. ∎

6.2.8 Proposition
In the $\leq$ ordering, every set of type schemes has a lower bound.

Proof: It is easy to see that for every type scheme σ

$$\forall\alpha.\alpha \leq \sigma$$

∎

6.3 Type systems

6.3.1 The traditional type system for pure ML

We assume that we have a fixed set of constants Q, and for each $c \in Q$ its type $\kappa(c)$, built only with type constants and arrows, is known.

The system depicted in Figure 6.1 will serve as a reference type system for ML [DM82, CDDK86, KTU89, KTU94]. We shall use the symbol $\vdash_{\mathrm{ML}}$

$$\text{(CON)} \qquad \vdash c : \kappa(c) \quad \text{for } c \in K$$

$$\text{(VAR)} \qquad E(x:\sigma) \vdash x : \sigma$$

$$\text{(ABS)} \qquad \frac{E(x:\tau) \vdash M : \rho}{E \vdash \lambda x.M : \tau \to \rho}$$

$$\text{(APP)} \qquad \frac{E \vdash M : \tau \to \rho \quad E \vdash N : \tau}{E \vdash MN : \rho}$$

$$\text{(GEN)} \qquad \frac{E \vdash M : \tau}{E \vdash M : \forall\alpha.\tau} \qquad \alpha \notin FV(E)$$

$$\text{(INST)} \qquad \frac{E \vdash M : \forall\alpha.\tau}{E \vdash M : \tau[\rho/\alpha]}$$

$$\text{(LET)} \qquad \frac{E \vdash M : \sigma \quad E(x:\sigma) \vdash N : \tau}{E \vdash \textbf{let } x = M \textbf{ in } N : \tau}$$

Figure 6.1: A reference ML type system, $\vdash_{ML}$

to denote derivability in this system.

The simplest way to extend this system with subtyping is by adding the subsumption rule

$$\text{(SUB)} \qquad \frac{E \vdash M : \tau \quad \tau \leq \rho}{E \vdash M : \rho}$$

In the sequel by $\vdash_{\mathrm{ML}_\leq}$ we shall understand the system $\vdash_{\mathrm{ML}}$ with the subsumption rule.

We shall say that a derivation is *normal* if subsumption rule is applied only to variables and constants.

6.3.1 Lemma
If $E \vdash_{\mathrm{ML}_\leq} M : \tau$ then there is a normal derivation ending with this judgement.

6.3.2 An alternative type system for ML

Kfoury et. al, in [KTU89, KTU94] suggest an alternative (equivalent) type inference system for ML, which is better suited for complexity studies:[2]

$$\begin{array}{lcl}
\text{(CON)} & \vdash c : \kappa(c) \quad \text{for } c \in K & \\[2ex]
\text{(VAR)} & (x : \sigma) \vdash x : \tau \qquad \text{for } \sigma \preceq \tau & \\[2ex]
(\mathit{ABS}) & \dfrac{E(x : \tau) \vdash M : \rho}{E \vdash \lambda x.M : \tau \to \rho} & \\[2ex]
(\mathit{APP}) & \dfrac{E \vdash M : \tau \to \rho \quad E \vdash N : \tau}{E \vdash MN : \rho} & \\[2ex]
(\mathit{LET}) & \dfrac{E \vdash M : \rho \quad E(x : \forall\vec{\alpha}.\rho) \vdash N : \tau}{E \vdash \mathbf{let}\ x = M\ \mathbf{in}\ N : \tau} & \alpha
\end{array}$$

Figure 6.2: An alternative type system for ML, $\vdash_{ML^*}$

6.3.2 Proposition
For every term M, it is typable in $\vdash_{\mathrm{ML}^\star}$ iff it is typable in $\vdash_{\mathrm{ML}}$.

For proof, cf. [CDDK86, KTU90].

Subtyping can be added here by replacing the instance relation in the axiom with the subtyping relation defined in the section 6.2.3...

$$E(x : \sigma) \vdash x : \tau \qquad \text{if} \vdash \sigma \leq \tau$$

...and modifying an axiom for constants:

$$\text{(CON)} \qquad \vdash c : \tau \qquad \text{if} \vdash \kappa(c) \leq \tau$$

We shall denote derivability in the resulting system by the symbol $\vdash_{\mathrm{ML}^\star_\leq}$.

6.3.3 Theorem
For every environment E term M, and open type τ, if $\vec{\alpha} \subseteq FV(\tau) - FV(E)$ then

$$E \vdash_{\mathrm{ML}_\leq} M : \forall\vec{\alpha}.\tau \text{ iff } E \vdash_{\mathrm{ML}^\star_\leq} M : \tau$$

In other words for every term M, it is typable in $\vdash_{\mathrm{ML}^\star_\leq}$ iff it is typable in $\vdash_{\mathrm{ML}_\leq}$.

[2]This system is called $\vdash^*$ in [KTU90]

Proof: The right-to-left implication becomes obvious when we observe that $\vdash_{\mathrm{ML}^\star_\leq}$ can be viewed as a subset of $\vdash_{\mathrm{ML}_\leq}$, and that under given assumption generalization over $\vec{\alpha}$ is allowed in $\vdash_{\mathrm{ML}_\leq}$.

The proof of the left-to-right implication may proceed by induction on derivation; the only difference from the Proposition 6.3.2 lies in the rule (SUB):

$$\text{(SUB)}\quad \frac{E \vdash M : \forall\vec{\alpha}.\,\tau \quad \forall\vec{\alpha}.\,\tau \leq \forall\vec{\beta}.\,\tau'}{E \vdash M : \forall\vec{\beta}.\,\tau'}$$

Because of the normalization theorem for $\leq$, it is sufficient to consider the cases when $\forall\vec{\alpha}.\,\tau \preceq \forall\beta.\,\tau'$ and when $\forall\vec{\alpha}.\,\tau \leq_\sigma \forall\beta.\,\tau'$.

In the first case it follows that $\tau' \equiv \tau[\vec{\rho}/\vec{\alpha}]$. By the induction assumption we have that

$$E \vdash_{\mathrm{ML}^\star_\leq} M : \tau$$

and want to conclude that

$$E \vdash_{\mathrm{ML}^\star_\leq} M : \tau[\vec{\rho}/\vec{\alpha}].$$

It is easy to see that

$$E[\vec{\rho}/\vec{\alpha}] \vdash_{\mathrm{ML}^\star_\leq} M : \tau[\vec{\rho}/\vec{\alpha}]$$

but since α's cannot be possibly free in E, we have that $E[\vec{\rho}/\vec{\alpha}] = E$.

Now consider the case when

$$\vdash_\sigma \forall\vec{\alpha}.\,\tau \leq \forall\beta.\,\tau'$$

It is easy to prove that in this case $\vec{\alpha} = \vec{\beta}$ and $\vdash_\sigma \tau \leq \tau'$ with τ, τ' open . A routine check that in this case if $E \vdash_{\mathrm{ML}^\star_\leq} M : \tau$ then $E \vdash_{\mathrm{ML}^\star_\leq} M : \tau'$ is left to the reader. ∎

6.3.4 Lemma
Let M be an arbitrary term, x a free variable, occurring k times ($k \geq 1$) in M, and let $N = \lambda x_1 \ldots x_k.M'$, where M' is a term obtained from M by replacing subsequent occurrences of x with $x_1, \ldots, x_k$ respectively. Then M is typable iff N is, or, more precisely

1. *If $E(x:\sigma) \vdash M : \tau$ then $E \vdash N : \rho_1 \to \cdots \to \rho_k \to \tau$ for some $\rho_1, \ldots, \rho_k$ such that $\sigma \leq \rho_i$ for $i = 1, \ldots, k$.*
2. *If $E \vdash N : \rho_1 \to \cdots \to \rho_k \to \tau$, then there is σ such that $E(x:\sigma) \vdash M : \tau$.*

Proof: Ad 1. Consider the derivation of $E(x:\sigma) \vdash M : \tau$. For ρ_i we take the type assigned by an axiom instance to i-th occurrence of x:

$$E(x:\sigma) \vdash x : \rho_i, \qquad \vdash \sigma \leq \rho_i.$$

It is easy to see that

$$E(x_1 : \rho_1, \ldots, x_k : \rho_k) \vdash M' : \tau.$$

From this, the thesis follows.

Ad 2. Let σ be a lower bound of the set $\{\rho_1, \ldots, \rho_k\}$ (cf. Prop. 6.2.8). If $E \vdash N : \rho_1 \to \cdots \to \rho_k \to \tau$, then also

$$E(x_1 : \rho_1, \ldots, x_k : \rho_k) \vdash M' : \tau.$$

The axiom instance for each x_i must look like

$$E(x_1 : \rho_1, \ldots, x_k : \rho_k) \vdash x_i : \rho_i$$

But then also

$$E(x : \sigma) \vdash x : \rho_i$$

since $\sigma \leq \rho_i$ for all i. Thus

$$E(x : \sigma) \vdash M : \tau.$$

∎

6.3.3 The canonical form of ML terms

Let us say that a term M is in *canonicalform* if it is of the form:

$$\begin{array}{lllll} M \equiv & \textbf{let} & x_1 & = N_1 & \textbf{in} \\ & \textbf{let} & x_2 & = N_2 & \textbf{in} \\ & & \vdots & & \\ & \textbf{let} & x_{n-1} & = N_{n-1} & \textbf{in} \\ & \textbf{let} & x_n & = N_n & \textbf{in}\ N_{n+1} \end{array}$$

for some $n \geq 0$, and where $N_1, \ldots, N_{n+1}$ do not contain **let**

6.3.5 Lemma ([KTU90])
Let M be an arbitrary term. We can construct a term M' in work-space logarithmic in $|M|$ such that:

1. *M' is in canonical form,*
2. *For every environment E and type τ, $E \vdash M : \tau$ iff $E \vdash M' : \tau$.*

6.3.6 Lemma
Let M be a term in canonical form, τ an open type and E an environment in which all types are closed. If there is a derivation D whose last assertion is $E \vdash M : \tau$, then there is a derivation D' with last assertion $E \vdash M : \tau$ and in which every environment type scheme is either closed (i.e. has no free type variables) or is a type.

Proof: Let $M = \textbf{let}\ x_1 = N_1\ \textbf{in} \ldots \textbf{let}\ x_n = N_n\ \textbf{in}\ N_{n+1}$, where N_i contain no **let**'s. We shall proceed by induction on n.

- If $n = 0$, then $M = N_1$ and in derivation occur, apart from the type schemes from E, only types.
- For the induction step, consider a term of the form

  $$M' \equiv \textbf{let}\ x = N\ \textbf{in}\ M,$$

  where M is in canonical form and contains at most n **let**'s, and N contains none. The last rule in D must have been

  $$\frac{E \vdash N : \rho \quad E(x : \forall\vec{\alpha}.\rho) \vdash N : \tau}{E \vdash \textbf{let}\ x = N\ \textbf{in}\ M : \tau}$$

  Where $\vec{\alpha} = \mathrm{FV}(\tau)$, since all type schemess in E are closed. By the induction assumption, M has a derivation where every environment type scheme either closed or a type, thus so has M'.

∎

6.3.4 Example

The following example illustrates an important difference between ML and $\mathrm{ML}_{\leq}$:

Assume we have two type constants i, r (one can think of them as representing for example *int* and *real*), with $i \leq r$, an atomic constant *pi* of type r and a functional constant *round* of type $r \to i$. Now consider the following term:

$$\textbf{let}\ t \ = \ (\lambda f.\lambda x.f\ (f\ x))\ \textbf{in}\ t\ \mathit{round}\ \mathit{pi}$$

This term is not typable in ML but is typable in $ML_{\leq}$. In fact all its normal typings mention only monomorphic types. One of this typings is presented in Fig. 6.3 Here, the context in which t is used, has 'forced' inferring a monomorphic type for it, even though its definition allows to infer an universal type, e.g. $\forall\alpha.\alpha \to \alpha$.

This example illustrates consequences of the fact that $ML_{\leq}$ has no principal types property, and shows that this type system is in a way 'not compositional', whereas for ML, the following holds:

Assumptions:
r and i are atomic type constants with $i \leq r$.

pi is an object constant of type r

$round$ is an object constant of type $r \rightarrow i$

$E_{f,x} = \{f : r \rightarrow i, x : r\}$

$E_t = \{t : (r \rightarrow i) \rightarrow r \rightarrow i\}$

Derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{E_{f,x} \vdash f : r \rightarrow i \qquad \dfrac{E_{f,x} \vdash f : r \rightarrow r \quad E_{f,x} \vdash x : r}{E_{f,x} \vdash fx : r}}{E_{f,x} \vdash f(fx) : i}}{f : r \rightarrow i \vdash \lambda x.f(fx) : r \rightarrow i}}{\vdash \lambda f.\lambda x.f(fx) : (r \rightarrow i) \rightarrow r \rightarrow i} \qquad \dfrac{\dfrac{E_t \vdash t : (r \rightarrow i) \rightarrow r \rightarrow i \quad \vdash round : r \rightarrow i}{E_t \vdash t\ round : r \rightarrow i} \qquad E_t \vdash pi : r}{E_t \vdash t\ round\ pi : i}}{\vdash \lambda f.\lambda x.f(fx) : (r \rightarrow i) \rightarrow r \rightarrow i}}{\vdash\ \textbf{let}\ t = \lambda f.\lambda x.f(fx)\ \textbf{in}\ \text{t round pi : i}}$$

Figure 6.3: A type derivation in $ML_{\leq}$

6.3.7 Proposition
For any terms N_1, N_2

$$E \vdash_{\mathrm{ML}} \textbf{\textit{let}}\ x = N_1\ \textbf{\textit{in}}\ N_2 : \tau$$

iff

$$E(x : \sigma) \vdash_{\mathrm{ML}} N_2 : \tau$$

where σ is a principal type for N_1.

6.4 Subtyping and Semi-Unification

6.4.1 Semi-unification

The Semi-Unification Problem (SUP) can be formulated as follows: An instance Γ of SUP is a set of pairs of open types. A substitution S is a *solution* of $\Gamma = \{(t_1, u_1), \ldots, (t_n, u_n)\}$ iff there are substitutions $R_1, \ldots, R_n$ such that

$$R_1(S(t_1)) = S(u_1), \ldots, R_n(S(t_n)) = S(u_n)$$

The problem is to decide, whether given instance has a solution.

A variation of this problem including subtyping can be formulated by redefining solution of Γ as follows: S is a *sub-solution* of Γ iff there are substitutions $R_1, \ldots, R_n$ such that

$$R_1(S(t_1)) \leq S(u_1), \ldots, R_n(S(t_n)) \leq S(u_n)$$

The Semi Sub-Unification Problem (SSUP) is to decide whether given instance has a sub-solution.

6.4.1 Proposition ([KTU93])
SUP is undecidable.

6.4.2 Corollary
SSUP is undecidable.

6.4.2 Acyclic semi-unification

An instance Γ of semi-unification is *acyclic* if for some $n \geq 1$, there are integers $r_1, \ldots, r_n$ and $n + 1$ disjoint sets of variables, $V_0, \ldots, V_n$, such that the pairs of Γ can be placed in n columns (possibly of different height; column i contains r_i pairs):

$$\begin{array}{cccc}
(t^{1,1},u^{1,1}) & (t^{2,1},u^{2,1}) & \cdots & (t^{n,1},u^{n,1}) \\
(t^{1,2},u^{1,2}) & (t^{2,2},u^{2,2}) & \cdots & (t^{n,2},u^{n,2}) \\
\vdots & \vdots & \ddots & \vdots \\
(t^{1,r_1},u^{1,r_1}) & (t^{2,r_2},u^{2,r_2}) & \cdots & (t^{n,r_n},u^{n,r_n})
\end{array}$$

where:

$$\begin{array}{rcl}
V_0 & = & \mathrm{FV}(t^{1,1}) \cup \cdots \cup \mathrm{FV}(t^{1,r_1}) \\
V_i & = & \mathrm{FV}(u^{i,1}) \cup \cdots \cup \mathrm{FV}(u^{i,r_i}) \cup \\
 & & \cup \mathrm{FV}(t^{i+1,1}) \cup \cdots \cup \mathrm{FV}(t^{i+1,r_{i+1}}) \text{ for } 1 \leq i < n \\
V_n & = & \mathrm{FV}(u^{n,1}) \cup \cdots \cup \mathrm{FV}(u^{n,r_n})
\end{array}$$

The set of terms with variables from V_i is also called *zone i*.

The Acyclic Semi-Unification Problem (ASUP) is the problem of deciding whether given acyclic instance has a solution.

Here again, subtyping can be introduced to yield an Acyclic Semi-Sub-Unification problem: whether a sub-solution of given acyclic instance exists.

6.4.3 Proposition ([KTU90])
ASUP is DEXPTIME-complete.

6.5 From ML$_\leq$ to ASSUP

First we have to make some assumption about constants in our language. For purpose of this section we assume there is a finite number of constants $c_1, \ldots, c_p$ and that the type of the constant c_i is $\tau(c_i)$. In fact, as the next section points out, one can make some more modest assumptions about constants.

Given a closed term M in canonical form we will construct an instance Γ_M of ASSUP such that M is typable iff Γ_M has a sub-solution. We shall assume the following notational conventions about the term M:

1. The **let**-bound variables are taken from the set $\{x_0, x_1, \ldots\}$.

2. The λ-bound variables are taken from the set $\{y_0, y_1, \ldots\}$.

3. Every variable is bound exactly once

The instance Γ_M of ASSUP will be written with variables all in the set:

$$\begin{aligned} V &= \{\beta_i \mid i \in \omega\} \cup \{\gamma_i \mid i \in \omega\} \cup \{\delta_i \mid i \in \omega\} \cup \\ &\cup \{\beta_i^{(j)} \mid i, j \in \omega\} \cup \{\gamma_i^{(j)} \mid i, j \in \omega\} \cup \{\kappa_i^{(j)} \mid i, j \in \omega\} \end{aligned}$$

We think of V as a set of type meta-variables, which is why we do not include α's in V, type expressions being written exclusively using type variables named by α's.

Suppose there is a derivation D which assigns a type to M. We think of β_i as standing for the assumed type of x_i before it is discharged in D (by an application of the rule LET), whereas $\beta_i^{(j)}$ stands for the type assigned to the j-th occurrence of x_i in M; γ_i stands for the assumed type of y_i before it is discharged (by an application of rule ABS), while $\gamma_i^{(j)}$ stands for the type assigned to the j-th occurrence of y_i in M. Finally $\kappa_i^{(j)}$ stands for the type of the j-th occurrence of the constant c_i. We include the δ's in V as auxiliary variables, which will be needed in the construction of Γ_M.

Before setting up Γ_M we construct an system Δ_M of type inequalities. The definition of Δ_M is by induction on M.

Let M be in canonical form, and $M_1, M_2, \ldots, M_n$ be all the subterms of M such that, for $k = 1, \ldots, n$, if M_k is not an object variable, then $M_k = (M_i M_j)$ or $(\lambda y.M_i)$ or $(\textbf{let}\ x = M_i\ \textbf{in}\ M_j)$ for some $i \neq j$ and $i, j \in \{1, 2, \ldots, k-1\}$. The set $\{M_1, M_2, \ldots, M_n\}$ mentions all occurrences of the same subterm, i.e., we may have $M_i = M_j$ for $i \neq j$. Observe that $M = M_n$.

Definition of Δ_k for $k = 1, \ldots, n$:

Simultaneously with Δ_k we define a type expression t_k with variables in V, by induction on $k = 1, \ldots, n$:

1. If M_k is the j-th occurrence of x_i in M, then set $\Delta_k = \emptyset$ and $t_k = \beta_i^{(j)}$. (We number the occurrences of x_i in M with $0, 1, 2, \ldots$, starting from the left end of M. The binding occurrence $\cdots$ **let** $x_i = \cdots$ is not counted.)

2. If M_k is the j-th occurrence of y_i in M, then set $\Delta_k = \{\gamma_i \leq \gamma_i^{(j)}\}$ and $t_k = \gamma_i^{(j)}$

3. If M_k is the j-th occurrence of c_i in M, then set $\Delta_k = \{\tau(c_i) \leq \kappa_i^{(j)}\}$ and $t_k = \kappa_i^{(j)}$

4. If $M_k = (M_i M_j)$ then set $\Delta_k = \Delta_i \cup \Delta_j \cup \{t_i \leq t_j \to \delta\}$ and $t_k = \delta$, where δ is a fresh auxiliary variable.

5. If $M_k = (\lambda y_i\ M_j)$ then set $\Delta_k = \Delta_j$ and $t_k = \gamma_i \to t_j$.
6. If $M_k = (\textbf{let}\ x_i = M_j\ \textbf{in}\ M_\ell)$ then set $\Delta_k = \Delta_j \cup \Delta_\ell \cup \{t_j \leq \beta_i\}$ and $t_k = t_\ell$.

Instead of Δ_n and t_n, we also write Δ_M and t_M. ■

Let $\mathcal{T}$ be the set of open types over variables named by α's and S be a substitution $S : V \to \mathcal{T}$. We say that S *satisfies* Δ_M ($S \models \Delta_M$) if $S(t) \leq_\tau S(u)$ for every inequality $t \leq u$ in Δ_M.

If $\tau \in \mathcal{T}$ and $\mathrm{FV}(\tau) = \{\alpha_1, \dots, \alpha_n\}$, then we write $\forall\bar{\varphi}.\,\tau$ to mean $\forall\vec{\alpha}.\tau$.

6.5.1 Lemma

Let M be a closed term in canonical form, and τ a type. The judgement $\vdash M : \tau$ is derivable iff there is a substitution S such that:

1. $S \models \Delta_M$,
2. $S(t_M) = \tau$,
3. $\forall\bar{\varphi}.\,S(\beta_i) \leq S(\beta_i^{(j)})$ *for all relevant i and j.*

Proof: Consider a derivation of $\vdash M : \tau$ and assume that all subterms of M are numbered as in the definition of Δ_M. This means that for every $1 \leq k \leq n$ there exists a type τ_k, and an environment E_k such that:

1. $E_k \vdash M_k : \tau_k$
2. $\mathrm{dom}(E_k) = FV(M_k)$ (in particular $E_n = \emptyset$)
3. if $M_k = M_i M_j$ then $E_i = E_j = E_k$
4. if $M_k = \lambda y_i.M_j$ then $E_j = E_k(y_i : \rho_i)$ for some ρ_i
5. if $M_k = (\textbf{let}\ x_i = M_j\ \textbf{in}\ M_\ell)$ then $E_j = E_k$ and $E_\ell = E_k(x_i : \forall\bar{\varphi}.\,\tau_j)$

It is easy to check that a substitution S such that $S(\beta_i) = \tau_j$ (where j is the number of the term at the right side of the defining occurrence of x_i), $S(\gamma_i) = \rho_i$ for all relevant i and such that its value for the superscripted variables is determined by types in respective axiom instances, fulfills all conditions of the thesis. The only unobvious case is the **let** expression, so let us have a closer look at it:

$$\frac{E \vdash M_j : \tau_j \quad E(x_i : \forall\bar{\varphi}.\,\tau_j) \vdash M_\ell : \tau_\ell}{E \vdash \textbf{let}\ x_i = M_j\ \textbf{in}\ M_l : \tau_\ell}$$

Now, remembering that in this case $\Delta_k = \Delta_j \cup \Delta_\ell \cup \{t_j \leq \beta_i\}$ and $t_k = t_\ell$, observe that $S(\beta_i) = S(t_j)$, thus $S \models \Delta_k$.

For the converse implication, assume that S fulfills all conditions of the lemma and define

1. $E_n = \emptyset$,
2. if $M_k = M_i M_j$ then $E_i = E_j = E_k$
3. if $M_k = \lambda y_i.M_j$ then $E_j = E_k(y_i : S(\gamma_i))$,
4. if $M_k = (\mathbf{let}\ x_i = M_j\ \mathbf{in}\ M_\ell)$ then $E_\ell = E_k(x_i : \forall\bar{\varphi}.\, S(\beta_i))$ and $E_j = E_k$

Now, it is easy to prove by induction on the structure of M that for all $1 \leq k \leq n$

$$E_k \vdash_{\mathrm{ML}_\leq} M_k : S(t_k)$$

Thus, by the Lemma 6.3.3[3]

$$E_k \vdash M_k : S(t_k)$$

Again, the interesting case is **let**:

$$\dfrac{\dfrac{E_k \vdash M_j : S(\tau_j) \quad S(\tau_j) \leq S(\beta_i)}{E_k \vdash M_j : S(\beta_i)} \qquad E_k(x_i : \forall\bar{\varphi}.\, S(\beta_i)) \vdash M_\ell : S(\tau_\ell)}{E_k \vdash \mathbf{let}\ x_i = M_j\ \mathbf{in}\ M_l : S(\tau_\ell)}$$

∎

6.5.2 Lemma
The set of inequalities Δ_M can be decomposed as follows:

$$\begin{array}{rcl} \Delta_M & = & \Delta_{N_1} \cup \{t_{N_1} \leq \beta_1\} \cup \\ & & \Delta_{N_2} \cup \{t_{N_2} \leq \beta_2\} \cup \\ & & \quad\vdots \\ & & \Delta_{N_n} \cup \{t_{N_n} \leq \beta_n\} \cup \\ & & \Delta_{N_{n+1}} \end{array}$$

where for $i = 1, \ldots, n+1$:

1. $\mathrm{FV}(\Delta_{N_i} \cup \{t_{N_i}\}) \ \cap\ \{\beta_k \mid k \in \omega\} \ =\ \emptyset$,
2. $\mathrm{FV}(\Delta_{N_i} \cup \{t_{N_i}\}) \ \cap\ \{\beta_k^{(\ell)} \mid k, \ell \in \omega\} \ \subseteq\ \{\beta_k^{(\ell)} \mid k < i$ *and* $\ell \in \omega\}$,

and for $1 \leq i < j \leq n+1$:

3. $\mathrm{FV}(\Delta_{N_i} \cup \{t_{N_i}\}) \ \cap\ \mathrm{FV}(\Delta_{N_j} \cup \{t_{N_j}\}) \ \subseteq\ \{\beta_k^{(\ell)} \mid k < i$ *and* $\ell \in \omega\}$.

[3]remember that in this section $\vdash$ stands for $\vdash_{\mathrm{ML}^\star_\leq}$

Proof: This follows immediately from our conventions for naming variables and the definition of Δ_M. ∎

Definition of Γ_M:

Consider the decomposition of Δ_M as given in Lemma 6.5.2. For each of the $n+1$ lines in this decomposition we define a single inequality $T_i \leq U_i$, where $i = 1, \ldots, n+1$.

For $i = 1, \ldots, n$, if $\Delta_{N_i} = \{t_1 = u_1, \ldots, t_p = u_p\}$ where the t's and u's are type expressions over V, then:

$$\begin{aligned} T_i &= (\delta_1 \to \delta_1) \to \cdots \to (\delta_p \to \delta_p) \to (\delta_{p+1} \to \delta_{p+1}) \\ U_i &= (t_1 \to u_1) \to \cdots \to (t_p \to u_p) \to (\beta_i \to t_{N_i}) \end{aligned}$$

where the δ's are fresh auxiliary variables.

If $\Delta_{N_{n+1}} = \{t_1 = u_1, \ldots, t_q = u_q\}$ where the t's and u's are type expressions over V, then:

$$\begin{aligned} T_{n+1} &= (\delta_1 \to \delta_1) \to \cdots \to (\delta_q \to \delta_q) \\ U_{n+1} &= (t_1 \to u_1) \to \cdots \to (t_q \to u_q) \end{aligned}$$

where the δ's are fresh auxiliary variables.

Define the instance Γ'_M (not yet Γ_M) as:

$$\begin{aligned} \Gamma'_M = \; & \{T_1 \leq U_1\} \cup \\ & \{T_2 \leq U_2\} \cup \{\beta_1 \leq \beta_1^{(j)} \mid \beta_1^{(j)} \in \mathrm{FV}(\Delta_M)\} \cup \\ & \quad \vdots \\ & \{T_{n+1} \leq U_{n+1}\} \cup \{\beta_n \leq \beta_n^{(j)} \mid \beta_n^{(j)} \in \mathrm{FV}(\Delta_M)\} \end{aligned}$$

Γ'_M is "almost" acyclic, but not quite, with each line in this decomposition corresponding to a column. If Γ'_M is not acyclic, it can only happen because there is some i, $1 \leq i \leq n$, and a variable $\beta_i^{(j)}$ such that:

$$\beta_i^{(j)} \notin \mathrm{FV}(U_{i+1}) \qquad \text{and} \qquad \beta_i^{(j)} \in \mathrm{FV}(U_{i+m})$$

for some $m \geq 2$. For each such $\beta_i^{(j)}$, we replace the single inequality

$$\{\beta_i \leq \beta_i^{(j)}\}$$

by the following m inequalities:

$$\{\beta_i \leq \delta_1,\ \delta_1 \leq \delta_2,\ \ldots\ ,\ \delta_{m-1} \leq \beta_i^{(j)}\}$$

where the δ's are fresh auxiliary variables. Note that this irregularity may not happen to γ-variables. The resulting instance of semi-subunification is the desired Γ_M.

6.5.3 Lemma
Γ'_M is an instance of semi-subunification such that:

1. *If $S : V \to \mathcal{T}$ is a solution of Γ'_M in the sense of semi-subunification, then $\vdash M : \tau$ where $S(t_M) = \tau$.*

2. *If $\vdash M : \tau$ for some open type τ, then there is $S : V \to \mathcal{T}$ which is a solution of Γ'_M in the sense of semi-subunification such that $S(t_M) = \tau$.*

Proof: If $S : V \to \mathcal{T}$ is a substitution, it is also easy to check that $S \models \Delta_{N_i} \cup \{\beta_i = t_{N_i}\}$ iff S is a solution of $\{T_i \leq U_i\}$ in the sense of semi-unification, for $i = 1, \ldots, n$. Likewise, $S \models \Delta_{N_{n+1}}$ iff S is a solution of $\{T_{n+1} \leq U_{n+1}\}$ in the sense of semi-unification. Hence, by Lemma 6.5.1 and the definition of Γ'_M (not Γ_M), we conclude that both parts of the present lemma are true. ∎

6.5.4 Corollary
Γ_M is an acyclic instance such that:

1. *If $S : V \to \mathcal{T}$ is a solution of Γ_M in the sense of semi-subunification, then $\vdash M : \tau$ where $S(t_M) = \tau$.*

2. *If $\vdash M : \tau$ for some open type τ, then there is $S : V \to \mathcal{T}$ which is a solution of Γ_M in the sense of semi-subunification such that $S(t_M) = \tau$.*

Proof: It is easily checked that Γ_M is acyclic. On the other hand from the way Γ_M is obtained from Γ'_M it is obvious that every solution of Γ_M is a solution of Γ'_M and that every solution of Γ'_M can be naturally extended to a solution of Γ_M. ∎

6.5.5 Theorem
ML$_\leq$ *typability is* log*-space reducible to ASSUP.*

Proof: This is an immediate consequence of the preceding lemma. It is not difficult to check that all the intermediary steps in the transformation from M to Γ_M are carried out in constant or logarithmic work-space. ∎

6.6 From ASSUP to ML$_\leq$

6.6.1 Constraining terms

For every type variable α_i we introduce object variables w_i, v_i. We assume that for every type constant $\kappa \in Q$ we there is a constant c_κ of this type, and a constant $c_{(\kappa\to\kappa)}$ of type $\kappa \to \kappa$. Now, for every monomorphic type τ, we define a term M_τ and a context $C_\tau[\ ]$ with one hole simultaneously by induction on τ (bear in mind that $K = \lambda x.\lambda y.x$):

1. $M_\kappa = c_\kappa$
   $C_\kappa[\,] = c_{(\kappa\to\kappa)}[\,]$
2. $M_{\alpha_i} = v_i w_i$
   $C_{\alpha_i}[\,] = v_i[\,]$
3. $M_{\tau_1\to\tau_2} = \lambda x.K M_{\tau_2} C_{\tau_1}[x]$
   $C_{\tau_1\to\tau_2}[\,] = C_{\tau_2}[[\,]M_{\tau_1}]$

Intuitively, M_τ can be used wherever enforcing τ as a lower bound is needed. Dually, the context $C_\tau[\,]$ imposes τ as an upper bound for types of the term placed inside it. These intuitions are formalised in the following

6.6.1 Lemma
Let $\tau, \rho_1, \ldots, \rho_\ell, \rho_1^1, \ldots, \rho_\ell^1, \rho_1^2, \ldots, \rho_\ell^2$ be arbitrary types such that

$$\mathrm{FV}(\tau) \subseteq \{\alpha_1, \ldots, \alpha_\ell\}.$$

$$\rho_i^1 \leq \rho_i^2 \text{ for } 1 \leq i \leq \ell$$

Furthermore, let S be a substitution such that $\rho_i^1 \leq S(\alpha_i) \leq \rho_i^2$ for $1 \leq i \leq \ell$. Then for any term N and environment E such that

$$E \supseteq \{v_i : \rho_i^1 \to \rho_i^2, w_i : \rho_i \mid 1 \leq i \leq \ell\}$$

we have for every type τ'':

1. *If*
   $$E \vdash M_\tau : \tau''$$
   then $S(\tau) \leq \tau''$
2. *If*
   $$E \vdash C_\tau[N] : \tau''$$
   then[4] *there exists $\tau' \leq S(\tau)$ such that*
   $$E \vdash N : \tau'$$

Proof: By induction on τ:

- If $\tau = \kappa$ then $S(\tau) = \kappa$, $M_\tau = c_\kappa$ and $C_\tau[\,] = c_{(\kappa\to\kappa)}[\,]$. Obviously $E \vdash c_\kappa : \tau''$ implies $\kappa \leq \tau''$.

  For the proof of the second part, consider a generic derivation of a typing for $C_\kappa[N]$:

  $$\frac{E \vdash c_{(\kappa\to\kappa)} : \tau' \to \tau'' \quad E \vdash N : \tau'}{E \vdash c_{(\kappa\to\kappa)} N : \tau''}$$

  If $E \vdash c_{(\kappa\to\kappa)} : \tau' \to \tau''$ then $(\kappa \to \kappa) \leq \tau' \to \tau''$, i.e. $\tau' \leq \kappa \leq \tau''$.

[4]An alternative formulation: [...] then $E \vdash N : S(\tau)$

- If $\tau = \alpha_i$ then we have $S(\tau) = \rho_i$, $M_\tau = v_i w_i$, and $C_\tau[\ ] = v_i[\ ]$. Consider a generic type derivation M_τ:

  $$\frac{E \vdash v_i : \tau' \to \tau'' \quad E \vdash w_i : \tau'}{E \vdash v_i w_i : \tau''}$$

  In every such derivation, $\rho_i^1 \to \rho_i^2 \leq \tau' \to \tau''$ and $\rho_i \leq \tau'$. From this in turn it follows that $\tau' \leq \rho_i^1 \leq S(\tau) \leq \rho_i^2 \leq \tau''$ and $S(\tau) \leq \tau''$.

  Now consider a generic type derivation for $C\tau[N]$:

  $$\frac{E \vdash v_i : \tau' \to \tau'' \quad E \vdash N : \tau'}{E \vdash v_i N : \tau''}$$

  In every such derivation, $\tau' \leq \rho_i^1 \leq S(\tau) \leq \rho_i^2 \leq \tau''$, hence $\tau' \leq S(\tau) \leq \tau''$.

- If $\tau = \tau_1 \to \tau_2$ then we have $S(\tau) = S(\tau_1) \to S(\tau_2)$, $C_\tau[\,] = C_{\tau_2}[[\,]M_{\tau_1}]$, and $M_\tau = \lambda x.KM_{\tau_2}C_{\tau_1}[x]$.

  Consider a generic type derivation for M_τ (where E_x stands for $E(x : \tau_x)$:

  $$\cfrac{\cfrac{E_x \vdash K : \tau_2'' \to \tau_1'' \to \tau' \quad E_x \vdash M_{\tau_2} : \tau_2'' \quad E_x \vdash C_{\tau_1}[x] : \tau_1''}{E_x \vdash KM_{\tau_2}C_{\tau_1}[x] : \tau'}}{E \vdash \lambda x.KM_{\tau_2}C_{\tau_1}[x] : \tau_x \to \tau'}$$

  For every such derivation, all of the following conditions hold:

  1. $\tau_2'' \leq \tau'$ (a property of K)
  2. $S(\tau_2) \leq \tau_2''$ (by the induction assumption for M_{τ_2})
  3. $\tau_x \leq S(\tau_1)$ (since by the induction assumption for $C_{\tau_1}[\ ]$, there exists τ_1' such that $\tau_1' \leq S(\tau_1)$ and $E(x : \tau_x) \vdash x : \tau_1'$)

  It is easy to check that this conditions imply $S(\tau) \leq \tau_x \to \tau'$.

  Now consider a generic derivation for NM_{τ_1}

  $$\frac{E \vdash N : \tau_1' \to \tau' \quad E \vdash M_{\tau_1} : \tau_1'}{E \vdash NM_{\tau_1} : \tau'}$$

  By the induction assumption for M_{τ_1}, in every such derivation $S(\tau_1) \leq \tau_1'$. Now, by the induction assumption for C_{τ_2}, If $E \vdash C_{\tau_2}[NM_{\tau_1}] : \tau''$ then there exists τ' such that on top of the previous condition, also $\tau_1' \leq S(\tau_2)$. The conjunction of these two conditions implies in turn

  $$\tau_1' \to \tau' \leq S(\tau_1 \to \tau_2)$$

which completes the proof. ∎

6.6.2 Lemma
Let $\tau, \rho_1, \ldots, \rho_\ell$ be arbitrary types such that

$$\mathrm{FV}(\tau) \subseteq \{\alpha_1, \ldots, \alpha_\ell\}.$$

Furthermore, let S be a substitution such that $S(\alpha_i) = \rho_i$ for $1 \leq i \leq \ell$. Then for any term N and environment E such that

$$E \supseteq \{v_i : \rho_i \to \rho_i, w_i : \rho_i \mid 1 \leq i \leq \ell\}$$

we have

1. *$E \vdash M_\tau : S(\tau)$*
2. *If $E \vdash N : S(\tau)$, then there exists τ'' such that $E \vdash C_\tau[N] : \tau''$*

Proof: By induction on τ:

- If $\tau = \kappa$ then $S(\tau) = \kappa$, $M_\tau = c_\kappa$ and $C_\tau[\ ] = c_{(\kappa\to\kappa)}[\ ]$. Obviously $E \vdash c_\kappa : \kappa$.

  For the proof of the second part, note that the following derivation is correct for all $\kappa'' \geq \kappa$:

$$\frac{E \vdash c_{(\kappa\to\kappa)} : \kappa \to \kappa'' \quad E \vdash N : \kappa}{E \vdash c_{(\kappa\to\kappa)}N : \kappa''}$$

- If $\tau = \alpha_i$ then we have $S(\tau) = \rho_i$, $M_\tau = v_i w_i$, and $C_\tau[\ ] = v_i[\ ]$. The following type derivation shows that in fact $E \vdash M_{\alpha_i} : S(\alpha_i)$:

$$\frac{E \vdash v_i : \rho_i \to \rho_i \quad E \vdash w_i : \rho_i}{E \vdash v_i w_i : \rho_i}$$

  Now consider the following derivation for $C_\tau[N]$:

$$\frac{E \vdash v_i : \rho_i \to \rho_i \quad E \vdash N : \rho_i}{E \vdash v_i N : \rho_i}$$

- If $\tau = \tau_1 \to \tau_2$ then we have $S(\tau) = S(\tau_1) \to S(\tau_2)$, $C_\tau[\,] = C_{\tau_2}[[\,]M_{\tau_1}]$, and $M_\tau = \lambda x.KM_{\tau_2}C_{\tau_1}[x]$.

  Let $E_x = E(x : S(\tau_1))$. By the induction assumption for τ_1, there exists τ_1'' such that $E \vdash C_{\tau_1}[x] : \tau_1''$. It is easy to check that the following type derivation for M_τ is correct:

$$\frac{\dfrac{E_x \vdash K : S(\tau_2) \to \tau_1'' \to S(\tau_2) \quad E_x \vdash M_{\tau_2} : S(\tau_2) \quad E_x \vdash C_{\tau_1}[x] : \tau_1''}{E_x \vdash KM_{\tau_2}C_{\tau_1}[x] : S(\tau_2)}}{E \vdash \lambda x.KM_{\tau_2}C_{\tau_1}[x] : S(\tau_1) \to S(\tau_2)}$$

By the induction assumption, $E \vdash M_{\tau_1} : S(\tau_1)$, hence the following derivation for NM_{τ_1} is correct:

$$\frac{E \vdash N : S(\tau_1) \to S(\tau_2) \quad E \vdash M_{\tau_1} : S(\tau_1)}{E \vdash NM_{\tau_1} : S(\tau_2)}$$

Now, by the induction assumption for τ_2, there exists τ_2'' such that $E \vdash C_{\tau_2}[NM_{\tau_1}] : \tau_2''$. This completes the proof.

∎

6.6.2 The encoding

Consider an instance Γ of ASSUP. Without loss of generality we may assume that all the columns of Γ have an equal number of inequalities r.

Let type variables in zone i, for $i = 0, 1, \ldots, n$, be:

$$\alpha_{i,1},\ \alpha_{i,2},\ \ldots\ ,\ \alpha_{i,\ell_i}$$

for some $\ell_i \geq 1$, corresponding to which we introduce object variables:

$$v_{i,1}, w_{i,1},\ v_{i,2}, w_{i,2},\ \ldots,\ v_{i,\ell_i}, w_{i,\ell_i}$$

The notation M_τ and $C_\tau[\ ]$ introduced earlier relative to singly subscripted variables, α_i and v_i, w_i, is now extended to doubly subscripted variables, $\alpha_{i,j}$ and $v_{i,j}, w_{i,j}$. We can assume that all the zones have an equal number ℓ of variables, i.e.,

$$\ell\ =\ \ell_0\ =\ \ell_1\ =\ \cdots\ =\ \ell_n$$

With these assumptions about Γ, let us introduce some building blocks which shall be used in the construction of the term M_Γ:

In the sequel by $C_i^\ell[\ ]$ we shall mean the context

$$\lambda w_{i,1}.\ldots\lambda w_{i,\ell}.(\lambda v_{i,1}.\ldots\lambda v_{i,\ell}.[\ ])\ \underbrace{I\ldots I}_{\ell \text{ times}}$$

where $I = \lambda x.x$.

Define

$$N_1' = \lambda z.zM_{t^{1,1}}\ldots M_{t^{1,r}}$$
$$N_1 = C_0^\ell[N_1']$$

Further for $1 \leq j \leq r$, define

$$P_{1,j} = \lambda p_1\ldots\lambda p_l.x_ip_1\ldots p_l(\lambda y_1\ldots\lambda y_r.C_{u^{1,j}}[y_j])$$

and for $2 \leq i \leq n$ and $1 \leq j \leq r$

$$P_{i,j} = \lambda p_1 \ldots \lambda p_l . x_i p_1 \ldots p_l (\lambda z_1 \ldots \lambda z_r \lambda y_1 \ldots \lambda y_r . C_{u^{i,j}}[y_j])$$

and

$$\begin{aligned} N'_i &= \lambda z . z P_{i-1,1} \ldots P_{i-1,r} M_{t^{i,1}} \ldots M_{t^{i,r}} && (6.4) \\ N_i &= C_i^{\ell}[N'_i] && (6.5) \end{aligned}$$

Finally we define the term M_Γ as follows:

$$\begin{aligned} M_\Gamma \equiv\ & \textbf{let}\ x_1 = N_1\ \textbf{in} \\ & \textbf{let}\ x_2 = N_2\ \textbf{in} \\ & \quad \vdots \\ & \textbf{let}\ x_{n-1} = N_{n-1}\ \textbf{in} \\ & \textbf{let}\ x_n = N_n\ \textbf{in}\ N_{n+1} \end{aligned}$$

6.6.3 Soundness of the encoding

6.6.3 Lemma
If Γ has a sub-solution then M_Γ is typable.

Proof: Suppose Γ has a sub-solution S:

$$S = [\alpha_{i,j} := \rho_{i,j} \mid i = 0, \ldots, n,\ \text{and}\ j = 1, \ldots, \ell\,]$$

There are therefore substitutions $R_{i,j}$ such that: $R_{i,j}(S(t^{i,j})) \leq S(u^{i,j})$, for every $i = 1, \ldots, n$ and $j = 1, \ldots, r$. We shall show that M_Γ is typable. For $i = 1, \ldots, n+1$, define the environment E_i:

$$E_i = \{v_{i-1,j} : \rho_{i-1,j} \to \rho_{i-1,j}, w_{i-1,j} : \rho_{i-1,j} \mid 1 \leq j \leq \ell\}$$

For $i = 1, \ldots, n$ and $j = 1, \ldots, r$, by Lemma 6.6.2:

$$E_i \vdash M_{t^{i,j}} : S(t^{i,j})$$

and for every $i = 2, \ldots, n+1$ and $j = 1, \ldots, r$ there exists an open type $\psi_{i,j}$, such that

$$E_i(y_j : S(u^{i-1,j})) \vdash C_{u^{i-1,j}}[y_j] : \psi_{i,j}$$

We shall prove that $\vdash N_1 : \xi_1$, where:

$$\tau_1 = \rho_{0,1} \to \cdots \to \rho_{0,\ell} \to (S(t^{1,1}) \to \cdots \to S(t^{1,r}) \to \beta_1) \to \beta_1$$

where β_1 is a type variable.

Let
$$\chi_1 = S(t^{1,1}) \to \cdots \to S(t^{1,r}) \to \beta_1$$
The desired property of N_1 is easily seen from the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{E_1(z:\chi_1) \vdash z:\chi_1 \quad E_1(z:\chi_1) \vdash M_{t^{1,1}} : S(t^{1,1})}{\vdots}}{E_1(z:\chi_1) \vdash zM_{t^{1,1}} \ldots M_{t^{1,r}} : \beta_1}}{E_1 \vdash N_1' : \chi_1 \to \beta_1}}{\dfrac{\vdots}{\vdash C_1^{\ell}[N_1'] : \xi_1}}$$

Let the type $\xi_1^{(j)}$ be defined as follows:
$$\xi_1^{(j)} = R_{1,j}(\xi_1)$$
note that thus defined $\xi_1^{(j)}$ is an instance of $\forall\bar{\varphi}.\,\xi_1$.

Now we shall prove that for every j there exists $\psi_{i,j}$ such that
$$E_2(x_1 : \xi_1) \vdash P_{i,j} : R_{1,j}(\rho_{1,1}) \to \cdots R_{1,j}(\rho_{1,\ell}) \to \psi_{i,j}$$
In the sequel we shall call the type of $P_{i,j}$ thus inferred $\theta_{i,j}$.

Observe that there exists $\psi_{1,j}$ such that
$$\begin{array}{lcl} E_2 \vdash \lambda y_1 \ldots \lambda y_r.C_{u^{1,j}}[y_j] & : & R_{1,j}(S(t^{1,1})) \to \cdots \to R_{1,j}(S(t^{1,j-1})) \to \\ & & S(u^{1,j}) \to \\ & & R_{1,j}(S(t^{1,j+1})) \to \cdots \to R_{1,j}(S(t^{1,r})) \to \psi_{1,j} \end{array}$$
but, since $R_{1,j}(S(t^{1,j}) \leq S(u^{1,j})$, also
$$\begin{array}{lcl} E_2 \vdash \lambda y_1 \ldots \lambda y_r.C_{u^{1,j}}[y_j] & : & R_{1,j}(S(t^{1,1})) \to \cdots \to R_{1,j}(S(t^{1,j-1})) \to \\ & & R(S(t^{1,j})) \to \\ & & R_{1,j}(S(t^{1,j+1})) \to \cdots \to R_{1,j}(S(t^{1,r})) \to \psi_{1,j} \end{array}$$
thus in fact
$$E_2(x_1 : \forall\bar{\varphi}.\,\xi_1) \vdash P_{1,j} : \theta_{1,j}$$

Basing on this, by an argument similar to the one about N_1, it is not difficult to prove that
$$\begin{array}{lcl} x_1 : \xi_1 \vdash N_2 & : & \rho_{1,1} \to \cdots \to \rho_{1,\ell} \to \\ & & (\theta_{1,1} \to \cdots \to \theta_{1,r} \to \\ & & S(t^{2,1}) \to \cdots \to S(t^{2,r}) \to \beta_2) \to \beta_2 \end{array}$$

we shall call this type ξ_2.

Similarly, for $i = 3, \ldots, n+1$ and $j = 1, \ldots, r$, using the fact that $R_{i-1,j}(S(t^{i-1,j})) \leq S(u^{i-1,j})$, it is not difficult to also check that:

$$\{x_{i-1} : \forall\bar{\varphi}.\, \xi_{i-1}\} \vdash N_i : \xi_i$$

where

$$\begin{aligned}\xi_i &= \rho_{i-1,1} \to \cdots \to \rho_{i-1,\ell} \to \\ &\quad (\theta_{i-1,1} \to \cdots \to \theta_{i-1,r} \to S(t^{i,1}) \to \cdots \to S(t^{i,r}) \to \beta_i) \to \beta_i \\ \\ \theta_{i,j} &= R_{i,j}(\rho_{i,1}) \to \cdots \to R_{i,j}(\rho_{i,\ell}) \to \psi_{i,j}\end{aligned}$$

for some type $\psi_{i,j}$. Indeed, for all relevant i, j there exists $\psi_{i,j}$ such that

$$\begin{aligned}E_{i+1} \vdash \lambda z_1 \ldots \lambda z_r \lambda y_1 \ldots \lambda y_r.\ \ & C_{u^{1,j}}[y_j] : \\ & \theta_{i-1,1} \to \cdots \to \theta_{i-1,r} \to \\ & R_{i,j}(S(t^{i,1})) \to \cdots \to R_{i,j}(S(t^{i,j-1})) \to \\ & S(u^{i,j}) \to \\ & R_{i,j}(S(t^{i,j+1})) \to \cdots \to R_{i,j}(S(t^{i,r})) \to \\ & \psi_{i,j}\end{aligned}$$

but, since $R_{i,j}(S(t^{i,j}) \leq S(u^{i,j})$, also

$$\begin{aligned}E_{i+1} \vdash \lambda z_1 \ldots \lambda z_r \lambda y_1 \ldots \lambda y_r.\ \ & C_{u^{1,j}}[y_j] : \\ & \theta_{i-1,1} \to \cdots \to \theta_{i-1,r} \to \\ & R_{i,j}(S(t^{i,1})) \to \cdots \to R_{i,j}(S(t^{i,j-1})) \to \\ & R(S(t^{i,j})) \to \\ & R_{i,j}(S(t^{i,j+1})) \to \cdots \to R_{i,j}(S(t^{i,r})) \to \\ & \psi_{i,j}\end{aligned}$$

thus in fact

$$E_{i+1}(x_i : \forall\bar{\varphi}.\, \xi_i) \vdash P_{i,j} : \theta_{i,j}$$

and

$$\{x_{i-1} : \forall\bar{\varphi}.\, \xi_{i-1}\} \vdash N_i : \xi_i$$

∎

6.6.4 Completeness of the encoding

6.6.4 Lemma
If M_Γ is typable then Γ has a sub-solution, i.e. there exist substitutions $S, R_{1,1}, \ldots, R_{n,r}$ such that

$$R_{i,j}(S(t_{i,j})) \leq S(u_{i,j}) \text{ for } 1 \leq i \leq n, \quad 1 \leq j \leq r$$

Proof: Suppose that M_Γ is typable. This means that, for $i = 1, \ldots, n+1$, N_i is typable in an environment E_i of the form:

$$E_i \;=\; \{x_1 : \sigma_1, \ldots, x_{i-1} : \sigma_{i-1}\}$$

where $\sigma_1, \ldots \sigma_n$ are type schemes Although the only free variable in N_i is x_{i-1}, E_i must include a type assumption for every variable whose binding includes N_i in its scope. Note that $\emptyset = E_1 \subset \cdots \subset E_{n+1}$.

Let V_i denote the set of variables in zone i of Γ. Note that

$$FV(N_i \;=\; \begin{cases} V_0 & \text{if } i = 1 \\ V_{i-1} \cup \{x_{i-1}\} & \text{otherwise} \end{cases} \tag{6.6}$$

If $N_i = C_i^\ell[N_i']$ is typable in E_i, then there exists an environment E_i^v and types $\xi_i, \rho^1_{i,1}, \ldots, \rho^1_{i,\ell}, \rho^2_{i,1}, \ldots, \rho^2_{i,\ell}$ such that

$$\rho^1_{i,j} \;\leq\; \rho^2_{i,j} \tag{6.7}$$

$$E_i^v(v_{i,j}) \;=\; \rho^1_{i,j} \to \rho^2_{i,j} \tag{6.8}$$

$$E_i^v \;\vdash\; N_i : \xi_i \tag{6.9}$$

Take any S such that $\rho^1_{i,j} \leq S(\alpha_{i,j}) \leq \rho^1_{i,j}$ for $1 \leq i \leq n$, $1 \leq j \leq \ell$. The existence of such S follows from acyclicity of Γ and (6.6). Note that S and E_i^v satisfy assumptions of Lemma 6.6.1.

First let us focus on the term N_1. The type ξ_1 must be of the form

$$\xi_1 = \rho_{0,1} \to \cdots \to \rho_{0,\ell} \to (\tau_{1,1} \to \cdots \to \tau_{1,r} \to \varphi_1) \to \psi_1$$

where

$$S(t^{1,j}) \;\leq\; \tau_{1,j} \text{ for } 1 \leq j \leq r \tag{6.10}$$

$$\varphi_1 \;\leq\; \psi_1 \tag{6.11}$$

Now consider the term $P_{1,j}$. There exist an environment $E^y_{1,j} \supseteq E^v_1$ and a type $\psi_{1,j}$ such that

$$E^y_{1,j} \vdash C_{u^{i,j}} : \psi_{1,j}$$

$$E^y_{1,j} \vdash y : S(u_{1,j})$$

Since $P_{1,j}$ occurs in a context **let** $x_1 = N_1$ **in** $\ldots$, the occurrence of x_1 in it must be assigned a type ξ_1^j that is an instance of $\forall\bar{\varphi}.\, \xi_1$. Therefore by the Lemma 6.2.7, there exists a substitution $R_{1,j}$ such that

$$R_{1,j}(\xi_1) \leq \xi_1^j. \tag{6.12}$$

From this, it follows that ξ_1^j must be of the form

$$\xi_1^j = \rho^j_{0,1} \to \cdots \to \rho^j_{0,\ell} \to (\tau^j_{1,1} \to \cdots \to \tau^j_{1,r} \to \varphi^j_1) \to \psi^j_1$$

since x_1 occurs in $P_{1,j}$ in the context $x_1 p_1 \ldots p_\ell(\lambda y_1 \ldots \lambda y_r.C_{u^1,j}[y_j]$, by Lemma 6.6.1 we have

$$\tau^j_{1,j} \leq S(u^{1,j})$$

Since $\tau_{1,j}$ occurs positively in ξ_1, we have $R_{1,j}(S(t^{1,j}) \leq R_{1,j}(\tau_{1,j})$. In turn $R_{1,j}(\tau_{1,j}) \leq \tau^j_{1,j}$ by 6.12. From this inequalities, we conclude that

$$R_{1,j}(S(t^{1,j})) \leq S(u^{1,j}).$$

By the same token, for $2 \leq i \leq n+1$,[5] the type ξ_i must be of the form

$$\begin{aligned}\xi_i &= \rho_{i-1,1} \to \cdots \to \rho_{i-1,\ell} \to \\ &\quad (\theta_{i-1,1} \to \cdots \to \theta_{i-1,r} \to \tau_{i,1} \to \cdots \to \tau_{i,r} \to \varphi_i) \to \psi_i\end{aligned}$$

and for all j the occurrence of x_i in $P_{i,j}$ must be assigned a type of the form

$$\begin{aligned}\xi^j_i &= \rho^j_{i-1,1} \to \cdots \to \rho^j_{i-1,\ell} \to \\ &\quad (\theta^j_{i-1,1} \to \cdots \to \theta^j_{i-1,r} \to \tau^j_{i,1} \to \cdots \to \tau^j_{i,r} \to \varphi^j_i) \to \psi^j_i\end{aligned}$$

and there must be a substitution $R_{i,j}$ such that

$$R_{i,j}(\xi_i) \leq \xi^j_i. \tag{6.13}$$

From this and from the construction of $P_{i,j}$ we can again conclude that in fact $R_{i,j}(S(t^{i,j}) \leq R_{i,j}(\tau_{i,j}) \leq \tau^j_{i,j}$ and hence

$$R_{i,j}(S(t^{i,j})) \leq S(u^{i,j})$$

for all i and j. ∎

6.6.5 Lemma
ASSUP is log-*space reducible to* **ML**$_\leq$ *typability.*

6.6.6 Theorem
ASSUP is log-*space equivalent to* **ML**$_\leq$ *typability.*

Proof: It is easy to see that the constructions shown in the preceding lemmas can be indeed done using logarithmic workspace. ∎

[5]Note that there is no x_{n+1}, but there is N_{n+1}.

Chapter 7

ML and constrained quantification

Contents

Type systems for ML with subtyping which use standard ML type schemes (such as one presented in the previous Chapter) suffer from two important deficiencies: they don't have the principal types property, nor the let-expansion property. The first deficiency all but precludes their practical use in implementations with separate compilation (though they use in the systems such as Standard ML could be possible, if a little awkward). The second one makes the complexity analysis extremely difficult.

Smith [Smi91] proposes a type system with constrained type schemes, where subtype constraints are a part of both environment and quantifier binding (presented in Fig 7.1 below).

$$C, E(x:\sigma) \vdash x:\sigma$$

$$\frac{C, E(x:\tau) \vdash M:\rho}{C, E \vdash \lambda x.M : \tau \to \rho}$$

$$\frac{C, E \vdash M:\tau \to \rho \quad C, E \vdash N:\tau}{C, E \vdash MN:\rho}$$

$$\frac{C_1 \cup C_2, E \vdash M:\tau}{C_1, E \vdash M : \forall\vec{\alpha}\,\textbf{with}\,C_2.\tau} \qquad \alpha_i \notin FV(E)$$

$$\frac{C_1, E \vdash M : \forall\vec{\alpha}\,\textbf{with}\,C.\tau \quad C_1 \vdash C[\vec{\rho}/\vec{\alpha}]}{C_1, E \vdash M : \tau[\vec{\rho}/\vec{\alpha}]}$$

$$\frac{C, E \vdash M:\sigma \quad C, E(x:\sigma) \vdash N:\tau}{C, E \vdash \textbf{let}\ x = M\ \textbf{in}\ N:\tau}$$

$$C(\tau \le \tau') \vdash \tau \le \tau' \qquad\qquad C \vdash \tau \le \tau$$

$$\frac{C \vdash \tau \le \tau' \quad C \vdash \tau' \le \tau''}{C \vdash \tau \le \tau''}$$

$$\frac{\tau' \le \tau \quad \rho \le \rho'}{C \vdash \tau \to \rho \le \tau' \to \rho'}$$

$$\frac{C, E \vdash M:\tau \quad C \vdash \tau \le \rho}{C, E \vdash M:\rho}$$

Figure 7.1: A type system with bounded quantification for ML

The advantage of using systems with constrained type schemes lies in their principal types property, which was lacking from the system presented in the previous chapter.

However, instead of considering the original system of Smith, in this chapter we shall show how it can be refined and analyse the complexity of the resulting system. The reason we do so is that in our opinion the system of Smith still doesn't satisfactorily solve the problem of principal types: the instance relation (which is usually defined on the syntactical level) proposed in his work is of semantical nature. He doesn't give any hints as to its syntactical characterisation of this relation, and it would be indeed very difficult (if at all possible) to construct such characterisation. We define a slightly different instance relation, which is of syntactical nature. The benefit from this can be instantly seen in that the proof of the principal types property is simpler than the one of Smith. Moreover, we prove the let-expansion property which we then use in the complexity analysis of typability.

7.1 Constrained type schemes

7.1.1 Syntax and conventions

By a constraint set we mean a set of the form

$$C = \{\tau_1 \leq \rho_1, \ldots, \tau_n \leq \rho_n\}$$

where τ_i, ρ_i are types. We shall use letters B, C, D (possibly with indices) to denote constraint sets.

A constrained type scheme has the following form:

$$\forall \vec{\alpha} \, \mathbf{with} \, C.\tau$$

where C is a constraint set and τ is a type. We shall use the letter σ with sub- or superscripts to denote type schemes. We shall also write $\forall \vec{\alpha}.\, \tau$ as an abbreviation for $\vdash \forall \vec{\alpha} \, \mathbf{with} \, \{\}.\tau$.

7.1.2 Inequalities

In this section we introduce a system of inferring inequalities between type schemes, being an adaptation of the system from the previous chapter to constrained type schemes. Then we show an analogon of the Normalization Theorem (6.2.7) from the previous chapter.

The system derives judgements of the form $C \vdash \sigma \leq \sigma'$, where σ and σ' are constrained type schemes.

Axioms:

(const)	$\vdash \kappa_1 \leq \kappa_2$ for $\kappa_1 \leq_\kappa \kappa_2$
(id)	$\tau \leq \tau' \vdash \tau \leq \tau'$
(refl)	$\vdash \sigma \leq \sigma$
(inst0)	$\vdash \forall\vec{\alpha}.\,\tau \leq \forall\vec{\beta}\,\mathbf{with}\,D.\,\tau[\vec{\rho}/\vec{\alpha}] \quad \vec{\beta} \notin FV(\forall\vec{\alpha}.\,\tau)$

Rules:

$$(\text{inst})\quad \frac{B \cup D \vdash C[\vec{\rho}/\vec{\alpha}]}{B \vdash \forall\vec{\alpha}\,\mathbf{with}\,C.\tau \leq \forall\vec{\beta}\,\mathbf{with}\,D.\,\tau[\vec{\rho}/\vec{\alpha}]} \quad \begin{cases}\vec{\beta} \notin FV(\forall\vec{\alpha}\,\mathbf{with}\,C.\tau) \\ \vec{\alpha}, \vec{\beta} \notin FV(B)\end{cases}$$

$$(\rightarrow)\quad \frac{C \vdash \rho' \leq \rho \quad C \vdash \tau \leq \tau'}{C \vdash \rho \rightarrow \tau \leq \rho' \rightarrow \tau'}$$

$$(\forall)\quad \frac{B \cup C \vdash \tau \leq \tau'}{B \vdash \forall\vec{\alpha}\,\mathbf{with}\,C.\,\tau \leq \forall\vec{\alpha}\,\mathbf{with}\,C.\,\tau'}$$

$$(\text{trans})\quad \frac{C \vdash \sigma \leq \sigma_1 \quad C \vdash \sigma_1 \leq \sigma'}{C \vdash \sigma \leq \sigma'}$$

7.1.3 Normalization

In this section we show that the system described above enjoys normalization properties in a manner similar to the one described in the previous chapter.

We shall use the symbol $\vdash_\sigma$ for structural derivations, i.e. ones that do not use neither the axiom (inst0) nor the rule (inst).[1]

7.1.1 Definition (Relation $\preceq_B$)
Let B be a set of constraints. We shall say that

$$\forall\vec{\alpha}\,\mathbf{with}\,C.\tau \preceq_B \forall\vec{\beta}\,\mathbf{with}\,D.\,\tau[\vec{\rho}/\vec{\alpha}]$$

if all of the following conditions hold:

1. $B \cup D \vdash C[\vec{\rho}/\vec{\alpha}]$
2. $\beta_i \notin FV(\forall\vec{\alpha}\,\mathbf{with}\,C.\tau)$
3. $\vec{\alpha}, \vec{\beta} \notin FV(B)$

[1]We intentionally reuse a symbol from the previous chapter as its intuitive meaning remains unchanged.

It is easy to see that the relation defined above is reflexive and transitive.

7.1.2 Lemma
For every set of constraints B and type schemes $\sigma, \sigma_1, \sigma'$, if $B \vdash_\sigma \sigma \leq \sigma_1$ and $\sigma_1 \preceq_B \sigma'$ then there exists σ_2 such that

$$\sigma \preceq_B \sigma_2$$

$$B \vdash_\sigma \sigma_2 \leq \sigma'.$$

Proof: If $B \vdash_\sigma \sigma \leq \sigma_1$ then there exist types τ, τ_1 and a set of constraints C such that

$$\sigma \equiv \forall \vec{\alpha}\, \mathbf{with}\, C.\, \tau$$

$$\sigma_1 \equiv \forall \vec{\alpha}\, \mathbf{with}\, C.\, \tau_1$$

$$B \cup C \vdash_\sigma \tau \leq \tau_1$$

Thus

$$\sigma' \equiv \forall \vec{\beta}\, \mathbf{with}\, D.\, \tau_1[\vec{\rho}/\vec{\alpha}]$$

and

$$D \vdash C[\vec{\rho}/\vec{\alpha}],$$

hence

$$\vdash \forall \vec{\alpha}\, \mathbf{with}\, C.\tau \preceq_B \forall \vec{\beta}\, \mathbf{with}\, D.\, \tau[\vec{\rho}/\vec{\alpha}]$$

The latter is the desired σ_2, since

$$B \vdash_\sigma \forall \vec{\beta}\, \mathbf{with}\, D.\, \tau[\vec{\rho}/\vec{\alpha}] \leq \forall \vec{\beta}\, \mathbf{with}\, D.\, \tau_1[\vec{\rho}/\vec{\alpha}]$$

∎

7.1.3 Theorem (Normalization)
If $B \vdash \sigma \leq \sigma'$ then there exists σ_1 such that

$$\sigma \preceq_B \sigma_1$$

$$B \vdash_\sigma \sigma_1 \leq \sigma'$$

Proof: The proof carries over, *mutatis mutandis*, from the analogous theorem (6.2.7) in the previous chapter. ∎

7.1.4 Generalization

7.1.4 Definition
Let E be an environment, C a set of constraints and τ a (monomorphic) type. The set of generalizations of τ with respect to C and E (in symbols: $Gen(C, E, \tau)$) is the set of all pairs $(C_1, \forall\vec{\alpha}\,\mathbf{with}\,C_2.\tau)$ such that

(i) $\vec{\alpha} = (FV(\tau) \cup FV(C)) \setminus FV(E)$

(ii) $C_1 \vdash C_2$

(iii) $C_1 \cup C_2 = C$

7.1.5 Lemma
For every C, E, τ, if C is satisfiable then

(i) $Gen(C, E, \tau) \neq \varnothing$

(ii) If $(C_1, \sigma) \in Gen(C, E, \tau)$ *then* $C_1 \vdash \sigma \leq \tau$

Proof: Let $\vec{\alpha} = (FV(\tau) \cup FV(C)) \setminus FV(E)$. It is readily verified that

$$(C, \forall\vec{\alpha}\,\mathbf{with}\,C.\tau) \in Gen(C, E, \tau)$$

The second part of the lemma follows as a direct application of the rule (inst) from the previous section:

$$\frac{C_1 \vdash C_2}{C_1 \vdash \forall\vec{\alpha}\,\mathbf{with}\,C_2.\tau \leq \tau}$$

∎

7.2 Type systems

7.2.1 The system BC_2

In the sequel, by $\vdash$ we shall mean $\vdash_{BC_2}$.

7.2.1 Lemma

(i) if $C, E \vdash M : \sigma$ *and* $E' \supseteq E$ *then* $C, E' \vdash M : \sigma$

(ii) if $C \vdash \sigma \leq \sigma'$ *and* $C' \supseteq C$ *then* $C' \vdash \sigma \leq \sigma'$

(iii) if $C, E \vdash M : \sigma$ *and* $C' \supseteq C$ *then* $C', E \vdash M : \sigma$

Proof: Statements (i) and (ii) can be proved by routine induction on the derivation. Statement (iii) follows from (ii). ∎

$$\frac{C \vdash \sigma \leq \tau}{C, E(x:\sigma) \vdash x : \tau}$$

$$\frac{C, E(x:\tau) \vdash M : \rho}{C, E \vdash \lambda x.M : \tau \to \rho}$$

$$\frac{C, E \vdash M : \tau \to \rho \quad C, E \vdash N : \tau}{C, E \vdash MN : \rho}$$

$$\frac{C, E \vdash M : \tau \qquad (C', \sigma) \in Gen(C, E, \tau) \quad C', E(x:\sigma) \vdash N : \rho}{C', E \vdash \mathbf{let}\ x = M\ \mathbf{in}\ N : \rho}$$

Figure 7.2: Type inference system BC_2

7.2.2 Lemma
If

(i) C *is a satisfiable constraint set,*

(ii) $C, E \vdash M : \tau$,

(iii) $(C_1, \sigma) \in Gen(C, E, \tau)$

(iv) $C_1 \vdash \sigma \leq \rho$

then

$$C', E \vdash M : \rho$$

Proof: Let $\sigma = \forall \vec{\alpha}\ \mathbf{with}\ C_2.\, \tau$. By the Normalization Theorem (7.1.3) there exists σ_1 such that

$$\sigma \preceq_B \sigma_1$$

$$B \vdash \sigma_1 \leq \rho$$

From this the thesis follows. ∎

7.3 Inference algorithms

In the algorithm below E stands for usual environment mapping variabloes to types, C for sets of constraints and A for so called meta-environment i.e. a partial function from variables to triples (set of constraints, environment, type).

The function *Refresh* systematically renames type variables to fresh names (to avoid possible name clashes). The symbols ν, nu_1, nu_2 stand for fresh type variables.

The function *gen* checks the satisfiability of the set of constraints passed to it as the first argument and fails if it is unsatisfiable (causing the whole algorithm to fail). Otherwise it yields an argument of the appropriate generalization set (which is non-empty by Lemma 7.1.5).

$$
\begin{array}{rcl}
PCT(x, A) & = \textbf{if} & A(x) = (C, E, \tau) \textbf{ then } (C \cup \{\tau \leq \nu\}, E, \nu) \\
 & & \textbf{else } (\{\nu_1 \leq \nu_2\}, \{x : \nu_1\}, \nu_2) \\
PCT(MN, A) & = \textbf{let} & \\
 & & (C_1, E_1, \tau_1) = PCT(M, A) \\
 & & (C_2, E_2, \tau_2) = \mathit{Refresh}(PCT(N, A)) \\
 & & S = \mathit{Unify}(\{\alpha = \beta \mid x : \alpha \in E_1 \wedge x : \beta \in E_2\} \\
 & & \qquad\qquad \cup \{\tau_1 = \tau_2 \to \nu\}) \\
 & \textbf{in} & \\
 & & S(C_1 \cup C_2, E_1 \cup E_2, \nu) \\
PCT(\lambda x.M, A) & = \textbf{let} & (C, E, \tau) = PCT(M, A) \\
 & \textbf{in} & \textbf{if } (x : \rho) \in E \\
 & & \textbf{then } \ (C, E \setminus \{(x : \rho)\}, \rho \to \tau) \\
 & & \textbf{else } \ (C, E, \nu \to \tau) \\
PCT(\textbf{let } \ x = M \textbf{ in } N, A) & = \textbf{let} & \\
 & & (C_1, E_1, \tau_1) = PCT(M, A) \\
 & & A' = A[x \mapsto (C_1, E_1, \tau_1)] \\
 & & (C_2, E_2, \tau_2) = \mathit{Refresh}(PCT(N, A')) \\
 & & S = \mathit{Unify}(\{\alpha = \beta \mid x : \alpha \in E_1 \wedge x : \beta \in E_2\}) \\
 & & (C', \sigma_1) = \mathit{gen}(S(C_1 \cup C_2), S(E_1 \cup E_2), S(\tau_1)) \\
 & \textbf{in} & \\
 & & (C', S(E_1 \cup E_2), S(\tau_2))
\end{array}
$$

7.3.1 Soundness of the algorithm PCT

7.3.1 Definition
Let A be a meta-environment i.e. a (partial) map from variables to triples (set of constraints, environment, type). We say that A is *well-formed* if, for

every $x \in \mathit{dom}(A)$, $\mathit{dom}(A) \cap \mathit{dom}(E) = \varnothing$, where $(C, E, \tau) = A(x)$.

7.3.2 Definition

Let E be an environment, A a well-formed meta-environment, $\mathit{dom}(A) \cap \mathit{dom}(E) = \varnothing$. We say that an environment E^A is an extension of E with A (in symbols $E^A \in \mathit{Ext}(E, A)$) if

1. $E \subseteq E^A$
2. For every $x \in \mathit{dom}(A)$ there exists $(C', \sigma) \in \mathit{Gen}(A(x))$ such that $x : \sigma$ in E^A.

7.3.3 Theorem

If A is a well-formed meta-environment, $\mathit{PCT}(M, A) = (C, E, \tau)$ then

1. $\mathit{dom}(A) \cap \mathit{dom}(E) = \varnothing$
2. $C, E^A \vdash M : \tau$

for every E^A in $\mathit{Ext}(E, A)$.

Proof: By induction on M:

- If $M \equiv x$, $x \notin \mathit{dom}(A)$, then the first part of the lemma follows from the assumption that A is well-formed. As for the second part, we have:

$$\mathit{PCT}(x, A) = (\{\nu_1 \leq \nu_2\}, \{x : \nu_1\}, \nu_2)$$

$$\{\nu_1 \leq \nu_2\}, x : \nu_1 \vdash x : \nu_2$$

- If $M \equiv x$, $x \in \mathit{dom}(A)$, $A(x) = (C, E, \tau)$ then:

$$\mathit{PCT}(x, A) = (C \cup \{\tau \leq \nu\}, E, \nu)$$

  and for every $(C', \sigma) \in \mathit{Gen}(C, E, \tau)$

$$C, x : \sigma \vdash x : \tau$$

  thus

$$C \cup \{\tau \leq \nu\}, x : \sigma \vdash x : \tau.$$

- If $M \equiv \lambda y.N$, then let $(C, E_1, \tau_1) = \mathit{PCT}(N, A)$. By the induction assumption, we have

$$C, E_1^A \vdash N : \tau_1$$

  Now, if $E_1 = E(x : \rho)$ then

$$\frac{C, E^A(x : \rho) \vdash N : \tau_1}{C, E^A \vdash \lambda x.\, N : \rho \to \tau_1}$$

On the other hand, if x does not occur in E_1 then it occurs neither in N nor in A, hence

$$\frac{C, E^A(x:\nu) \vdash N : \tau_1}{C, E^A \vdash \lambda x.\, N : \nu \to \tau_1}$$

- If $M \equiv M_1 M_2$, then let

$$\begin{aligned}
&(C_1, E_1, \tau_1) = PCT(M, A)\\
&(C_2, E_2, \tau_2) = \mathit{Refresh}(PCT(N, A))\\
&S = \mathit{Unify}(\{\alpha = \beta \mid x : \alpha \in E_1 \wedge x : \beta \in E_2\}\\
&\qquad\qquad \cup \{\tau_1 = \tau_2 \to \nu\})
\end{aligned}$$

By the induction assumption, we have

$$C_1, E_1 \vdash M_1 : \tau_1$$

$$C_2, E_2 \vdash M_2 : \tau_2$$

thus

$$\frac{\begin{array}{c} S(C_1 \cup C_2), (S(E_1 \cup E_2))^A \vdash M_1 : S(\tau_2) \to S(\nu) \\ S(C_1 \cup C_2), (S(E_1 \cup E_2))^A \vdash M_2 : S(\tau_2) \end{array}}{S(C_1 \cup C_2), (S(E_1 \cup E_2))^A \vdash M_1 M_2 : S(\nu)}$$

- If $M \equiv \textbf{let } x = M_1 \textbf{ in } M_2$, then let

$$\begin{aligned}
&(C_1, E_1, \tau_1) = PCT(M_1, A)\\
&A' = A[x \mapsto (C_1, E_1, \tau_1)]\\
&(C_2, E_2, \tau_2) = \mathit{Refresh}(PCT(M_2, A'))\\
&S = \mathit{Unify}(\{\alpha = \beta \mid x : \alpha \in E_1 \wedge x : \beta \in E_2\})\\
&(C_1', \sigma_1) = \mathit{gen}(S(C_1 \cup C_2), S(E_1 \cup E_2), \tau_1)
\end{aligned}$$

By the induction assumption we have that A' is well-formed and

$$C_1, E_1^A \vdash M_1 : \tau_1$$

$$C_2, E_2^{A'} \vdash M_2 : \tau_2$$

Hence

$$S(C_1 \cup C_2), S(E_1 \cup E_2)^A \vdash M_1 : S(\tau_1)$$

$$S(C_1 \cup C_2), S(E_1 \cup E_2)^{A'} \vdash M_2 : S(\tau_2)$$

Thus by applying the typing rule for **let** we obtain

$$C', \vdash S(E_1 \cup E_2)^A \textbf{let } x = M_1 \textbf{ in } M_2 : S(\tau_2)$$

which we wanted to prove.

∎

7.3.4 Corollary
If M is a closed term, $PCT(M, \varnothing) = (C, E, \tau)$ then

$$C, E \vdash M : \tau$$

Proof: As the empty-meta-environment is obviously well-formed, this follows immediately from the previous theorem. ∎

7.3.2 Principal types

7.3.5 Lemma
Let

$$E = \{x_1 : \sigma_1, x_2 : \sigma_2, \dots x_n : \sigma_n\}$$
$$C, E \vdash M : \tau$$

with satisfiable C and, for i such that $x_i \in FV(M)$, $(x_i : \sigma_i) \in E$, let

(i) $C_i, E_i \vdash P_i : \tau_i$,

(ii) C_i *satisfiable,*

(iii) $(C'_i, \sigma_i) \in Gen(C_i, E_i, \tau_i)$,

(iv) $A(x_i) = (C_i, E_i, \tau_i)$.

Then:

$$PCT(M, A) = (C_0, E_0, \tau_0)$$
$$\forall (C'_0, \sigma_0) \in Gen(C_0, E_0, \tau_0).\ C'_0 \vdash \sigma_0 \leq \tau$$

Proof: By induction on M:

- If $M \equiv x_i$ then we have:

$$PCT(x_i, A) = (C_i \cup \{\tau \leq \nu\}, E_i, \nu)$$

$$Gen(C_i \cup \{\tau \leq \nu\}, E_i, \nu) = \{(\varnothing, \forall \vec{\alpha}\ \mathbf{with}\ \tau \leq \nu . \nu\}$$

and there exist $\vec{\pi}$ such that

$$\varnothing \vdash (\forall \vec{\alpha}\ \mathbf{with}\ \tau_i \leq \nu . \nu) \leq \tau_i[\vec{\pi}/\vec{\alpha}]$$

$$\sigma_i = \forall \vec{\alpha}\ \mathbf{with}\ C''_i . \tau_i$$
$$C \vdash \sigma_i \leq \tau_i[\vec{\pi}/\vec{\alpha}]$$
$$C \vdash \tau_i[\vec{\pi}/\vec{\alpha}] \leq \rho$$

from which the thesis follows.

- If $M \equiv \lambda y.N$, then $\rho = \rho_1 \to \rho_2$ and
$$C, E(y : \rho_1) \vdash N : \rho_2$$
for some monomorphic types ρ_1, ρ_2. Let $A' = A[y \mapsto (\varnothing, \{y : \rho_1\}, \rho_1)]$. By the induction assumption, we have:
$$PCT(N, A') = (C_0, E_0, \rho_0)$$
and for all $(C', \sigma') \in Gen(C_0, E_0, \rho_0)$, $C' \vdash \sigma' \leq \rho_2$. It is easy to see that
$$PCT(\lambda y.N, A) = (C_0, E_0^y, \tau_0 \to \rho_0)$$
where $(y : \tau_0) \in E_0$ and $E_0^y = E_0 \setminus \{y : \tau_0\}$.
By definition of Gen,
$$FV(\tau_0) \cap BV(\sigma') = \varnothing.$$
Hence, for every $(C_2, \sigma_2) \in Gen(C_0, E_0^y, \tau_0, \rho_0)$,
$$C_2 \vdash \sigma_2 \leq \rho_1 \to \rho_2.$$

- If $M \equiv M_1 M_2$, then
$$C, E \vdash M_1 : \rho \to \tau$$
$$C, E \vdash M_2 : \rho$$
for some monomorphic type ρ By the induction assumption, we have:
$$PCT(M_1, A) = (C_1, E_1, \tau_1) \quad (7.1)$$
$$\forall (C_1', \sigma_1) \in Gen(C_1, E_1, \tau_1). \quad C_1' \vdash \sigma_1 \leq \rho \to \tau \quad (7.2)$$
$$PCT(M_2, A) = (C_2, E_2, \tau_2) \quad (7.3)$$
$$\forall (C_2', \sigma_2) \in Gen(C_2, E_2, \tau_2). \quad C_2' \vdash \sigma_2 \leq \rho \quad (7.4)$$
It follows that $\tau_1 = \tau_1^1 \to \tau_1^2$. Let S be the most general unifier of the set
$$\{\alpha = \beta \mid x : \alpha \in E_1 \wedge x : \beta \in E_2\} \cup \{\tau_1 = \tau_2 \to \nu\}$$
Further, let
$$C_0 = S(C_1 \cup C_2)$$
$$E_0 = S(E_1 \cup E_2)$$
$$\tau_0 = S(\nu)$$
We have
$$PCT(M_1 M_2, A) = (C_0, E_0, \tau_0)$$
And for all $\sigma = \forall \vec{\alpha}\, \textbf{with}\, C_2'.\tau_0$ such that $(C', \sigma_0) \in Gen(C_0, E_0, \tau_0)$:
$$C' \vdash \sigma_0 \leq \tau$$

- If $M \equiv$ **let** $x = M_1$ **in** M_2, then

$$C, E \vdash M_1 : \rho_1$$

$$C, E(x : \sigma) \vdash M_2 : \rho_2$$

$$(C', \sigma) \in Gen(C, E, \rho_1)$$

for some monomorphic types ρ_1, ρ_2.

By the induction assumption and Theorem 7.3.3, we have:

$$\begin{array}{rcll} PCT(M_1, A) & = & (C_1, E_1, \tau_1) & (7.5) \\ \forall(C_1', \sigma_1) \in Gen(C_1, E_1, \tau_1). & & C_1' \vdash \sigma_1 \leq \rho_1 & (7.6) \\ C_1, E_1^A & \vdash & M_1 : \tau_1 & (7.7) \\ PCT(M_2, A[x \mapsto (C_1, E_1, \tau_1)]) & = & (C_2, E_2, \tau_2) & (7.8) \\ \forall(C_2', \sigma_2) \in Gen(C_2, E_2, \tau_2). & & C_2' \vdash \sigma_2 \leq \rho_2 & (7.9) \end{array}$$

Let S be the mgu of the set

$$\{\alpha = \beta \mid x : \alpha \in E_1 \wedge x : \beta \in E_2\}$$

Further, let

$$C_0 = S(C_1 \cup C_2)$$

$$E_0 = S(E_1 \cup E_2)$$

$$\tau_0 = S(\tau_2)$$

We have

$$PCT(M, A) = (C_0, E_0, \tau_0)$$

And for all $\sigma = \forall\vec{\alpha}\,\textbf{with}\,C_2'.\tau_0$ such that $(C', \sigma_0) \in Gen(C_0, E_0, \tau_0)$:

$$C' \vdash \sigma_0 \leq \tau$$

∎

7.3.6 Theorem
For every closed term M, if

$$C, E \vdash M : \tau$$

$$Q \models C$$

then

$$PCT(M, \varnothing) = (C_0, \varnothing, \tau_0)$$

$$\forall\vec{\alpha}\,\textbf{with}\,C_0.\tau_0 \leq \tau$$

Proof: IF M is closed and $C, E \vdash M : \tau$ for some E then also $C, \varnothing \vdash M : \tau$. Thus this theorem is just a special case of the previous lemma. ∎

7.4 Complexity issues

7.4.1 Lemma (let-expansion)
If $C, E \vdash \textbf{let } x = M \textbf{ in } N : \rho$ then $C, E \vdash N[M/x] : \rho$

Proof: The derivation of $C, E \vdash \textbf{let } x = M \textbf{ in } N : \rho$ must end with

$$\frac{C_0, E \vdash M : \tau \qquad (C, \sigma) \in \mathit{Gen}(C_0, E, \tau) \quad C, E(x : \sigma) \vdash N : \rho}{C, E \vdash \textbf{let } x = M \textbf{ in } N : \rho}$$

Consider an occurrence $x^{(j)}$ of x in N and an instance of start rule for it in the derivation of $C, E(x : \sigma) \vdash N : \rho$:

$$\frac{C^j \vdash \sigma \leq \tau^j}{C^j, E^j(x : \sigma) \vdash x : \tau^j}$$

Note that $C^j \supseteq C$ and $E^j \supseteq E$. Thus, by lemmas 7.2.2 and 7.2.1, we have

$$C^j, E^j \vdash M : \tau^j$$

Hence

$$C, E \vdash N[M/x] : \rho$$

∎

7.4.2 Lemma (let-contraction)
If x occurs in M, $C, E \vdash N[M/x] : \rho$ and $FV(M) \subseteq \mathit{dom}(E)$ then $C, E \vdash$ $\textbf{let } x = M \textbf{ in } N : \rho$.

Proof: Assume that x occurs $n > 0$ times in M and consider all occurrences x^j of the variable xv in M and corresponding occurrences of typing judgements for M in the derivation of $C, E \vdash N[M/x] : \tau$:

$$C^j, E^j \vdash M : \tau^j$$

Let $C^0 = \bigcup_{j=1}^n C^j$. Since $FV(M) \subseteq FV(E)$, we have for all j

$$C^0, E \vdash M : \tau^j$$

Let (C_0^0, σ_0) be an arbitrary element of $\mathit{Gen}(C^0, E, \tau_1)$. By Lemma 7.3.5, we have

$$C_0^0 \vdash \sigma_0 \leq \tau^j$$

Hence,

$$C_0^0, E(x : \sigma_0) \vdash N : \rho$$

∎

In the sequel we shall only consider terms where for every subterm of the form **let** $x = M$ **in** N, the variable x occurs in N. We can do it without loss of generality, for if x does not occur in N, the let-expression can be replaced by

$$(\lambda xy.y)MN$$

which is equivalent from both typability and semantical point of view. Besides, in a practical setting such 'degenerated' let-expressions are likely to be eliminated even before type analysis (cf. e.g. [PJ87]).

7.4.3 Definition
Let M be an ML-term. By its *expansion* we mean the pure λ-term $[M]^*$ defined as follows:

- $[x]^* = x$
- $[\lambda x.N]^* = \lambda x.[N]^*$
- $[NP]^* = [N]^*[P]^*$
- $[\textbf{let } x = N \textbf{ in } P]^* = [P[N/x]]^*$

7.4.4 Theorem (let elimination)
Let M be a closed ML-term. It is typable iff $[M]^*$ is typable.

Proof: This theorem follows directly from Lemmas 7.4.1 and 7.4.2. ∎

7.4.5 Theorem
If the set of type constants with atomic inequalities between them for a poset for which SSI is decidable in polynomial time, then for every closed ML-term M, typability of M can be decided in time bounded by $p(|M| * 2^d)$ where d is the maximum depth of let-nesting in M and p is a fixed polynomial.

Proof: Let M be a closed ML term. By the previous theorem, M is typable if and only if $[M]^*$ is typable. It is easy to see that the size of $[M]^*$ is bounded by $|M| * 2^d$ where d is the maximum depth of let-nesting in M. Thus we can construct a system of inequalities Σ_M of size $\mathcal{O}(|M| * 2^d)$ such that M is typable iff Σ_M is satisfiable. But satisfiability of Σ_M can be checked in polynomial time, from which the thesis follows. ∎

Chapter 8

Conclusions and future research

Having concluded the technical part, it is time to look back and summarize what have been done and what remains for the future.

First, using order-theoretic as well as logical methods we have proven that although the SSI problem is PSPACE hard in general, there is a wide class of posets, containing many interesting and important classes (amongst others: lattices, trees and absolute retracts) for which this problem can be solved in polynomial time. Moreover, we have gathered substantial evidence in favour of the conjecture stating that the relation between complexities of FLAT-SSI and SSI corresponds to the relation between nondeterministic and alternating complexity classes.

Further, we have analysed two families of type systems for ML with subtyping, concluding that for partial orders for which SSI can be decided in polynomial time typability in ML with subtyping has the same complexity as typability without subtyping.

Yet the subject is far from being exhausted. This work has led to several questions and problems which we consider important:

- For which posets is SSI tractable? We give a partial answer to this, but a complete classification is still lacking.
- Can Conjecture 5.1.1 be formally proved?
- What is the exact complexity of ASSUP?
- Is there a more efficient way to check typability in ML with subtyping than by doing let-expansion, or is the exponential jump in complexity inherent for ML?

We strongly feel that answer to this questions (possibly except of ASSUP complexity) are difficult and need much more research.

Bibliography

[AC91] Robert M. Amadio and Luca Cardelli. Subtyping recursive types. In *Conf. Rec. 18th ACM Symposium on Principles of Programming Languages*, pages 104–118, 1991.

[App89] Andrew W. Appel. Runtime tags aren't necessary. *Lisp and Symbolic Computation*, 2:153–162, 1989.

[ASU86] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools.* Addison-Wesley, Reading, MA, 1986.

[Bar77] Henk P. Barendregt. The type free lambda calculus. In J. Barwise, editor, *Handbook of Mathematical Logic*, pages 1091–1132. North-Holland, Amsterdam, 1977.

[Bar81] Henk P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics.* North-Holland, Amsterdam, 1981.

[Bar84] Henk P. Barendregt. Introduction to lambda calculus. *Nieuw Archief voor Wiskunde*, 4(2):337–372, 1984.

[BDG90] Jose Luis Balcazar, Josep Diaz, and Joaquim Gabarro. *Structural Complexity II.* Springer Verlag, 1990.

[BDG95] Jose Luis Balcazar, Josep Diaz, and Joaquim Gabarro. *Structural Complexity I.* Springer Verlag, 1995. Second edition.

[Ben92] Nick Benton. *Strictness Analysis of Lazy Functional Programs.* PhD thesis, University of Cambridge Computer Laboratory, December 1992.

[Ben93] Marcin Benke. Efficient type reconstruction in the presence of inheritance (extended abstract). In *Proc. Int. Symp. MFCS 1993.* Springer Verlag, 1993.

[Ben94] Marcin Benke. Efficient type reconstruction in the presence of inheritance. Technical Report TR94-10(199), Institute of Informatics, Warsaw University, December 1994.

[Ben95] Marcin Benke. Some complexity bounds for subtype inequalities. Technical Report TR95-20(220), Institute of Informatics, Warsaw University, December 1995.

[Ben96a] Marcin Benke. An algebraic characterization of typability in ML with subtyping. In Humboldt-Universität, editor, *L .Czaja, P. Starke, H.-D. Burkhard, M. Lenz (Eds.), Workshop Concurrency Specification & Programming 1996*, number 69 in Informatik-Bericht, 1996.

[Ben96b] Marcin Benke. An algebraic characterization of typability in ML with subtyping. Technical Report TR96-14(235), Institute of Informatics, Warsaw University, December 1996.

[Ben96c] Marcin Benke. A logical approach to complexity bounds for subtype inequalities. In Petr Hajek, editor, *Gödel'96: logical foundations of mathematics, computer science and physics — Kurt Gödels legacy; Brno, proceedings*, Lecture Notes in Logic, pages 195–204. Springer, 1996.

[BH92] James M. Boyle and Terence J. Harmer. A practical functional program for the CRAY X-MP. *Journal of Functional Programming*, 2(1):81–126, January 1992.

[BM92] Kim Bruce and John C. Mitchell. Per models of subtyping, recursive types and higher-order polymorphism. In *Conf. Rec. ACM Symp. Principles of Programming Languages*, 1992.

[BTCGS89a] Val Breazu-Tannen, Thierry Coquand, Carl A. Gunter, and Andre Scedrov. Inheritance and explicit coercion. In *Proc. IEEE Symp. on Logic in Computer Science*, pages 112–129, 1989.

[BTCGS89b] Val Breazu-Tannen, Thierry Coquand, Carl A. Gunter, and Andre Scedrov. Inheritance and explicit coercion: Preliminary report. In *Proc. 4th IEEE Symposium on Logic in Computer Science*, pages 112–129, 1989.

[BTCGS90] Val Breazu-Tannen, Thierry Coquand, Carl A. Gunter, and Andre Scedrov. Computing with coercions. In *Proc. ACM Conference on Lisp and Functional Programming*, pages 44–61, 1990.

[BTCGS91] Val Breazu-Tannen, Thierry Coquand, Carl A. Gunter, and Andre Scedrov. Inheritance as explicit coercion. *Information and Computation*, 93(1):172–221, 1991.

[BTM85] Val Breazu-Tannen and Albert R. Meyer. Lambda calculus with constrained types. In *Logics of Programs*, pages 23–40, Berlin, June 1985. Springer LNCS 193.

[Car84] Luca Cardelli. A semantics of multiple inheritance. In *Semantics of Data Types, International Symposium*, volume 173 of *Lecture Notes in Computer Science*, pages 51–68. Springer-Verlag, Berlin, Heidelberg, New York, 1984.

[Car88] Luca Cardelli. Structural subtyping and the notion of powertype. In *Proc 15th ACM Symp. Principles of Programming Languages*, pages 70–79, 1988.

[Car92] Luca Cardelli. Extensible records in a pure calculus of subtyping. Technical Report 81, DEC Systems Research Center, 1992.

[CCHO89] Peter Canning, William Cook, Walter Hill, and W. Olthoff. Interfaces for strongly-typed object-oriented programming. In *Proceedings 3rd Symposium on Object-Oriented Programming, Languages, and Systems*, 1989.

[CD80] Mario Coppo and Mariangiola 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.

[CDDK86] D. Clément, J. Despeyroux, T. Despeyroux, and G. Kahn. A simple applicative language: Mini-ML. In *Proc. ACM Conference on Lisp and Functional Programming*, pages 13–27, 1986.

[CGL92] Giuseppe Castagna, Ghiorgio Ghelli, and Giuseppe Longo. A calculus for overloaded functions with subtyping. In *1992 ACM Conf. Lisp and Functional Programming*, pages 182–192, 1992.

[CHC90] Cook, Hill, and Canning. Inheritance is not subtyping. In *Conf. Rec. 17th ACM Symp. Principles of Programming Languages*, 1990.

[Chu36] Alonzo Church. An unsolvable problem in elementary number theory. *Amer. J. Math.*, 58:345–363, 1936.

[Chu40] Alonzo Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.

[CS76] Ashok K. Chandra and Larry J. Stockmeyer. Alternation. In *17th Annual Symposium on Foundations of Computer Science*, pages 98–108, Houston, Texas, 25–27 October 1976. IEEE.

[Cur34] Haskell B. Curry. Functionality in combinatory logic. *Proc. Nat. Acad. Science USA*, 20:584–590, 1934.

[Dam84] Luis Manuel Martins Damas. *Type assignment in programming languages.* PhD thesis, University of Edinburgh, 1984.

[DB80] N.G. De Bruijn. A survey of the project Automath. In *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 579–607. Academic Press, 1980.

[DKM84] Cynthia Dwork, Paris Kanellakis, and John C. Mitchell. On the sequential nature of unification. *Journal of Logic Programming*, 1:35–50, 1984.

[DM82] Luis Damas and Robin Milner. Principal type-schemes for functional programs. In *Conf. Rec. ACM Symp. Principles of Programming Languages*, pages 207–211, 1982.

[DPR80] Dwight Duffus, W. Poguntke, and Ivan Rival. Retracts and the fixed point problem for finite partially ordered sets. *Canad. Math. Bull.*, 23:231–236, 1980.

[DR79] Dwight Duffus and Ivan Rival. Retracts of partially ordered sets. *J. Austral. Math. Soc. (Ser. A)*, 27:495–506, 1979.

[DR81] Dwight Duffus and Ivan Rival. A structure theory for ordered sets. *Discrete Mathematics*, 35:53–118, 1981.

[FM88] You-Chin Fuh and Prateek Mishra. Type inference with subtypes. In *Proceedings European Symposium on Programming*, pages 94–114, 1988.

[FM89] You-Chin Fuh and Prateek Mishra. Polymorphic subtype inference: closing the theory-practice gap. In *Proc. Theory and Practice of Software Development*, pages 167–184, March 1989.

[FM90] You-Chin Fuh and Prateek Mishra. Type inference with subtypes. *Theoretical Computer Science*, 73:155–176, 1990.

[Fre97] Alexandre Frey. Satisfying systems of subtype inequalities in polynomial space. To appear in Proceedings of Fourth International Static Analysis Symposium, 1997.

[GH91] Hugh Glaser and Peter Henderson. *Functional programming and software enginnering.* Butterworth, 1991.

[GM94] Carl A. Gunter and John C. Mitchell, editors. *Theoretical Aspects of Object-Oriented Programming: Types, Semantics, and Language Design.* The MIT Press, 1994.

[Gur83] Yuri Gurevich. Algebras of feasible functions. In *22ndIEEE Symp. Foundations of Computer Science*, pages 210–214, 1983.

[Gur84] Yuri Gurevich. Toward logic tailored for computational complexity. In M. Richter et al., editors, *Computation and Proof Theory*, Lecture Notes in Math. 1104, pages 175–216. Springer, Berlin/New York, 1984.

[Har93] Rachel Harrison. The use of functional languages in teaching computer science. *Journal of Functional Programming*, 3(1):67–75, January 1993.

[Hen88] Fritz Henglein. Type Inference and Semi-Unification. In *LISP*, pages 184–197, ACM Press, July 1988.

[Hen90] Fritz Henglein. A lower bound for full polymorphic type inference: Girard/reynolds typability is DEXPTIME-hard. Technical Report RUU-CS-90-14, Univerity of Utrecht, 1990.

[HF92] P. Hudak and J. Fasel. A gentle introduction to Haskell. *SIGPLAN Notices*, 27(5):Section T, 1992.

[Hin69] Roger Hindley. The principal type-scheme of an object in combinatory logic. *Transactions of the American Mathematical Society*, 146:29–60, 1969.

[Hin83a] Roger Hindley. The completeness theorem for typing λ-terms. *Theoretical Computer Science*, 22:1–17, 1983.

[Hin83b] Roger Hindley. Curry's type-rules are complete with respect to the f-semantics too. *Theoretical Computer Science*, 22:127–133, 1983.

[HM94] F. Henglein and H. Mairson. The complexity of type inference for higher-order typed lambda calculi. *J. Functional Programming*, 4(4):435–478, 1994.

[HM95] My Hoang and John C. Mitchell. Lower bounds on type inference with subtypes. In *Conf. Rec. ACM Symp. Principles of Programming Languages*, 1995.

[HP92] Martin Hoffman and Benjamin Pierce. An abstract view of objects and subtyping. Technical Report ECS-LFCS-92-226, University of Edinburgh, Department of Computer Science, August 1992.

[HR97] Fritz Henglein and Jakob Rehof. The complexity of subtype entailment for simple types. In *Proc. 12th IEEE Symposium on Logic in Computer Science*, pages 352–361, 1997.

[Hud92] P. et al. Hudak. Report on the programming language Haskell. *SIGPLAN Notices*, 27(5):Section R, 1992.

[Imm88] Neil Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*, 17, 1988.

[JBH93] Stef Joosten (ED.), Klaas Van Den Berg, and Gerrit Van Der Hoeven. Teaching functional programming to first-year students. *Journal of Functional Programming*, 3(1):49–65, January 1993.

[Jen92] T. P. Jensen. *Abstract Interpretation in Logical Form.* PhD thesis, Imperial College, University of London, November 1992. Available as DIKU Report 93/11 from DIKU, University of Copenhagen.

[Jim96] Trevor Jim. What are principal typings and what are they good for? In *Conf. Rec. ACM Symp. Principles of Programming Languages*, pages 42–53, 1996.

[JM88] Lalita A. Jategaonkar and John C. Mitchell. ML with extended pattern matching and subtypes. In *Proc. 1988 ACM Symposium on Lisp and Functional Programming*, pages 198–211, 1988.

[KM89] Paris C. Kanellakis and John C. Mitchell. Polymorphic unification and ML typing. In *Conf. Rec. ACM Symp. Principles of Programming Languages*, pages 105–115, 1989.

[KPS92] Dexter Kozen, Jens Palsberg, and Michael I. Schwartzbach. Efficient inference of partial types. Technical Report DAIMI PB-394, Computer science Dept., Aarhus University, April 1992.

[KR74] D. Kelly and Ivan Rival. Crowns, fences and dismantlable lattices. *Canad J. Math.*, 26:1257–1271, 1974.

[KTU89] Assaf J. Kfoury, Jerzy Tiuryn, and Paweł Urzyczyn. An analysis of ML typability. Technical Report 89-009, Boston University, 1989.

[KTU90] Assaf J. Kfoury, Jerzy Tiuryn, and Paweł. Urzyczyn. ML typability is Dexptime-complete. In *Proc. 15th Colloq. on Trees in Algebra and Programming*, pages 206–220. Springer LNCS 431, 1990.

[KTU93] Assaf J. Kfoury, Jerzy Tiuryn, and Paweł Urzyczyn. The undecidability of the semi-unification problem. *Information and Computation*, 102(1):83–101, January 1993.

[KTU94] Assaf J. Kfoury, Jerzy Tiuryn, and Paweł Urzyczyn. An analysis of ML typability. *Journal of the ACM*, 41(2):368–398, March 1994.

[Lan65] Peter J. Landin. A correspondence between ALGOL 60 and Church's lambda-notation: Parts I and II. *Communications of the ACM*, 8(2,3):89–101, 158–165, February and March 1965.

[Lan66] Peter J. Landin. The next 700 programming languages. *Communications of the ACM*, 9:157–166, 1966.

[LLS78] Richard E. Ladner, Richard J. Lipton, and Larry J. Stockmeyer. Alternating pushdown automata (preliminary report). In *19th Annual Symposium on Foundations of Computer Science*, pages 92–106, Ann Arbor, Michigan, 16–18 October 1978. IEEE.

[LM92] Patrick Lincoln and John C. Mitchell. Algorithmic aspects of type inference with subtypes. In *Conf. Rec. ACM Symp. Principles of Programming Languages*, pages 293–304, 1992.

[LMS95] G. Longo, K. Milsted, and S. Soloviev. A logic of subtyping. In *Proc. IEEE Symp. on Logic in Computer Science*, pages 292–299, 1995.

[LS82] H.R. Lewis and R. Statman. Unifiability is complete for co-NLOG-space. *Information Processing Letters*, 15(5):220–222, December 1982.

[Mai90] H.G. Mairson. Deciding ML typability is complete for deterministic exponential time. In *Proc. 17th ACM Symp. Principles of Programming Languages*, pages 382–401, January 1990.

[Mil84] Robin Milner. *The standard ML core language.* Dept Computer Science, University of Edinburgh, 1984.

[Mit83a] John C. Mitchell. The implication problem for functional and inclusion dependencies. *Information and Control*, 56:112–138, 1983.

[Mit83b] John C. Mitchell. Inference rules for functional and inclusion dependencies. In *Proc. 2nd ACM Conf. on Principles of Database Systems*, pages 58–69, March 1983.

[Mit84] John C. Mitchell. Coercion and type inference. In *Conf. Rec. ACM Symp. Principles of Programming Languages*, pages 175–185, 1984.

[Mit88] John C. Mitchell. Polymorphic type inference and containment. *Information and Computation*, 76(2/3):211–249, 1988. Reprinted in *Logical Foundations of Functional Programming,* ed. G. Huet, Addison-Wesley (1990) 153–194.

[Mit90a] John C. Mitchell. Toward a typed foundation for method specialization and inheritance. In *Proc. 17th ACM Symp. on Principles of Programming Languages*, pages 109–124, January 1990.

[Mit90b] John C. Mitchell. *Type Systems for Programming Languages.* Elsevier, 1990.

[Mit91] John C. Mitchell. Type inference with simple subtypes. *Journal of Functional Programming*, 1(3):245–285, July 1991.

[MMM91] John C. Mitchell, Sigurd Meldal, and Neel Madhav. An extension of standard ML modules with subtyping and inheritance. In *Conf. Rec. 18th ACM Symp. Principles of Programming Languages*, pages 270–278. ACM, January 1991.

[Mon81] B. Monjardet. Metrics on partially ordered sets — a survey. *Discrete Mathematics*, 35:173–184, 1981.

[MTH89] Robin Milner, Mads Tofte, and Robert Harper. *The Definition of Standard ML.* MIT Press, Cambridge, MA, 1989.

[NP95] Tobias Nipkow and Christian Prehofer. Type reconstruction for type classes,. *Journal of Functional Programming*, 5(2):201–224, April 1995.

[NR85] P. Nevermann and Ivan Rival. Holes in ordered sets. *Graphs and Combinatorics*, (1):339–350, 1985.

[OB88] A. Ohori and P. Buneman. Type inference in a database language. In *Proc. ACM Symp. Lisp and Functional Programming Languages*, pages 174–183, July 1988.

[OL96] Martin Odersky and Konstantin Läufer. Putting type annotations to work. In *Conf. Rec. ACM Symp. Principles of Programming Languages*, pages 54–67, 1996.

[Pal94] Jens Palsberg. Efficient inference of object types. In *IEEE Symp. Logic in Computer Science*, 1994.

[Pap94] Christos H. Papadmitriou. *Computational complexity.* Addison-Wesley, 1994.

[Pau91] Lawrence C. Paulson. *ML for the Working Programmer.* Cambridge University Press, 1991.

[PJ87] Simon L. Peyton Jones. *The Implementation of Functional Programming Languages.* Prentice-Hall International, 1987.

[PS94] Jens Palsberg and Michael I. Schwartzbach. *Object-oriented Type Systems.* Wiley, 1994.

[PT92] Benjamin C. Pierce and David N. Turner. Object-oriented programming without recursive types. Technical Report ECS-LFCS-92-225, LFCS, University of Edinburgh, August 1992.

[PT93] Benjamin C. Pierce and David N. Turner. Object-oriented programming without recursive types. In *Proc 20th ACM Symp. Principles of Programming Languages*, pages 299–312, 1993.

[PT94] Benjamin C. Pierce and David N. Turner. Simple type-theoretic foundations for object-oriented programming. *Journal of Functional Programming*, 4(2):207–247, April 1994.

[PT95] Vaughn Pratt and Jerzy Tiuryn. Satisfiability of inequalities in a poset. Technical Report TR 95-15(215), Institute of Informatics, Warsaw University, 1995.

[PT96] Vaughn Pratt and Jerzy Tiuryn. Satisfiability of inequalities in a poset. *Fundamenta Informaticae*, 28(1,2):165–182, 1996.

[Qui83] A. Quillot. An application of the Helly property to the partially ordered sets. *J. Comb. Theory (A)*, 35:185–198, 1983.

[Qui85] A. Quillot. On the Helly property working as a compactness criterion for graphs. *J. Comb. Theory (A)*, 40:186–193, 1985.

[Rém89] Didier Rémy. Typechecking records and variants in a natural extension of ML. In *Conf. Rec. 16th ACM Symposium on Principles of Programming Languages*, pages 77–88, 1989.

[Rey80] John C. Reynolds. Using category theory to design implicit conversions and generic operators. In N.D. Jones, editor, *Semantics-Directed Compiler Generation*, pages 211–2580. Springer LNCS 94, Berlin, 1980.

[Riv76] Ivan Rival. A fixed point theorem for finite partially ordered sets. *J. Combinatorial Theory (A)*, 21:309–318, 1976.

[Riv82] Ivan Rival. *Ordered Sets.* Reidel, Dordrecht/Boston/London, 1982.

[Rob65] J. Alan Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12:23–41, 1965.

[Rod94a] Michael S. Roddy. Cores and retracts. *Order*, 11(1):1–10, 1994.

[Rod94b] Michael S. Roddy. Fixed points and products. *Order*, 11(1):11–14, 1994.

[RS94] Aleksander Rutkowski and Bernd S. Schröder. Retractability and the fixed point property for products. *Order*, 11(4):353–359, 1994.

[Smi91] Geoffrey S. Smith. Polymorphic type inference for languages with overloading and subtyping. Ph.D Thesis 91-1230, Department of Computer Science, Cornell University, Ithaca, NY 14853-7501, August 1991.

[Smi94] Geoffrey S. Smith. Principal type schemes for functional programs with overloading and subtyping. *Science of Computer Programming*, 23:197–226, 1994.

[Sta85] Richard Statman. Logical relations and the typed lambda calculus. *Information and Control*, 65:85–97, 1985.

[Sta88] Ryan Stansifer. Type inference with subtypes. In *Conf. Rec. 15th ACM Symposium on Principles of Programming Languages*, pages 88–97, 1988.

[Tiu92] Jerzy Tiuryn. Subtype inequalities. In *Proc. 7th IEEE Symposium on Logic in Computer Science*, pages 308–315, 1992.

[Tiu95] Jerzy Tiuryn. Equational axiomatization of bicoercibility for polymorphic types. In Ed. P.S. Thiagarajan, editor, *Proc. 15th Conference Foundations of Software Technology and Theoretical Computer Science*, volume 1026 of *Lecture Notes in Computer Science*, pages 166–179. Springer Verlag, 1995.

[Tiu96] Jerzy Tiuryn. A sequent calculus for subtyping polymorphic types. In A. Szał as W. Penczek, editor, *Proc. 21-st International Symposium on Mathematical Foundations of Computer Science*, volume 1113, pages 135–155. Springer Verlag, 1996.

[TU96] Jerzy Tiuryn and Paweł Urzyczyn. The subtyping problem for second-order types is undecidable. In *Proc. 11th IEEE Symposium on Logic in Computer Science*, 1996.

[Tur35] Alan M. Turing. On computable numbers with an application to the Entscheidungsproblem. *Proc. London Math. Soc. (2)*, 42:230–265, 1935.

[TW93] Jerzy Tiuryn and Mitchell Wand. Type reconstruction with recursive types and atomic subtyping. In M.-C. Gaudel and J.-P. Jouannaud, editors, *TAPSOFT'93: Theory and Practice of Software Development, Proc. 4th Intern. Joint Conf. CAAP/FASE*, pages 686–701. Springer-Verlag, 1993. LNCS 668.

[Tys88] Jerzy Tyszkiewicz. Complexity of type inference in finitely typed lambda calculus. Master's thesis, University of Warsaw, 1988.

[VS91] D. M. Volpano and Geoffrey S. Smith. On the complexity of ML typability with overloading. In *Proc. ACM Conf. Functional Programming and Computer Architecture*, 1991.

[Wan84] M. Wand. A types-as-sets semantics for Milner-style polymorphism. In *Proc. 11th ACM Symp. on Principles of Programming Languages*, pages 158–164, January 1984.

[Wan86] M. Wand. Finding the source of type errors. In *Conf. Rec. ACM Symp. Principles of Programming Languages*, pages 38–43, January 1986.

[Wan87a] M. Wand. Complete type inference for simple objects. In *Proc. IEEE Symp. on Logic in Computer Science*, pages 37–44, 1987. Corrigendum in *Proc. IEEE Symp. on Logic in Computer Science,* page 132, 1988.

[Wan87b] M. Wand. A simple algorithm and proof for type inference. *Fundamenta Informaticae*, 10:115–122, 1987.

[Wan88] M. Wand. Corrigendum: Complete type inference for simple objects. Proc. IEEE Symp. on Logic in Computer Science, 1988.

[Wan89a] M. Wand. Type inference for record concatenation and multiple inheritance. In *Proc. IEEE Symp. on Logic in Computer Science*, pages 92–97, 1989. To appear in *Information and Computation.*

[Wan89b] Mitchell Wand. Type inference for objects with instance variables and inheritance. Technical Report NU-CCS-89-2, Northeastern University College of Computer Science, Boston, MA, February 1989.

[Wan89c] Mitchell Wand. Type inference for record concatenation and multiple inheritance. In *Proc. 4th IEEE Symposium on Logic in Computer Science*, pages 92–97, 1989.

[Wan91] Mitchell Wand. Type inference for record concatenation and multiple inheritance. *Information and Computation*, 93:1–15, 1991.

[WB89] P. Wadler and S. Blott. How to make *ad-hoc* polymorphism less *ad hoc*. In *16th ACM Symposium on Principles of Programming Languages*, pages 60–76, 1989.

[Wel94] J. Wells. Typability and type checking in the second-order λ-calculus are equivalent and undecidable. In *Proc. 9th Ann. IEEE Symp. Logic in Comput. Sci.*, pages 176–185, 1994.

[Wel95] J. Wells. The undecidability of Mitchell's subtyping relation. Technical report, Computer Sci. Dept., Boston University, December 1995.

[WO89] M. Wand and Patrick O'Keefe. On the complexity of type inference with coercion. In *Proc. ACM Conf. Functional Programming and Computer Architecture*, 1989.

[Xia92] Weiqun Xia. Fixed point property and formal concept analysis. *Order*, 9(3):255–264, 1992.