A Semantic Characteriation of Descriptive Tpe Sstems

Mário Florido Luís Damas

Technical Report Series: DCC-98-6

Departamento de Ciência de Computadores – Faculdade de Ciências

&

Laboratório de Inteligência Artificial e Ciência de Computadores

Universidade do Porto

Rua do Campo Alegre, 823 4150 Porto, Portugal

Tel: +351+2+6078830 – Fax: +351+2+6003654

http://www.ncc.up.pt/fcup/DCC/Pubs/treports.html

A Semantic Characterization of Descriptive Type Systems

Mário Florido, Luís Damas
Computer Science Department, University of Porto,
R. do Campo Alegre, 823, 4150 Porto, Portugal,
E-mail: {amf,luis}@ncc.up.pt

Abstract

Traditionally type systems are defined and then proved to be sound with respect to some well chosen semantics. We argue that the definition of the systems should come from their semantics and not the opposite. We use constrained type systems to define necessary conditions for types to be descriptions of terms. We then prove the soundness with respect to those conditions of the Curry and ML type systems.

1 Introduction

The theoretical foundations of type systems was established by Alonzo Church, [Chu40], with the typed λ-calculus and by Haskell Curry in [Cur34]. Those two works led to two different views of types: the former corresponds to prescriptive types, where terms have a semantic in typed domains, and the latter to descriptive types, where terms and types have different semantics, and type systems define a relation of two different semantic entities.

In this paper we will follow the descriptive approach. Types and terms belong to distinct semantic domains and type systems define a relation between them. There are types semantically sound which the type system does not infer. Suppose we have a domain of basic types (integers, reals, etc) and an order relation between them which is given by semantic inclusion. Suppose that $int \sqsubseteq real$ i.e. if a value is an integer then it is a real. Then we have $int \rightarrow real$ as a semantic valid type for $\lambda x.x$, because the identity function, when applied to a integer returns a real. But in the simple type system $int \rightarrow real$ is not a type for $\lambda x.x$. More generally $\lambda x.x$ has got a valid semantic type $\alpha \rightarrow \beta$ if $\alpha \sqsubseteq \beta$. From all types $\alpha \rightarrow \beta$ in those conditions the one which better describes $\lambda x.x$ is $\alpha \rightarrow \alpha$, and that is precisely the principal type for the identity function in the simple type system. This is not a coincidence and we show that it is implied by a semantic relation between types and terms stronger than soundness and that we call strong soundness.

The constraints of the form $\alpha \sqsubseteq \beta$ are familiar in the context of *constrained type systems*, [FM88], [AW93], [PS96]. We define some obvious constrained type systems to capture a general semantic relation between terms and types defined only by the descriptive character of types. Then we define proper type systems which correspond to different ways of solving the set of constraints derived by the most general systems. Those proper type systems are defined by the goal of the system itself. For example monomorphism will define a set of solutions and second order polymorphism will define another set of solutions. The strong soundness property is then defined as follows: if a type τ is derived for a term M then τ has to be proper with respect to M. The definition of proper varies with the goal of the system: proper types in a monorphic system are different from proper types in a polymorphic system, but we argue that for any system types have to be proper with respect to the goal of that system.

We study strong soundness in one monomorphic system, the Curry type system for the lambda calculus, and one polymorphic system, the Damas-Milner type system. We show that the Curry and the Damas-Milner type systems define proper types, i.e, are strongly sound.

2 Descriptive Types

Our term language is the λ-calculus, [Bar84]. For the semantics of the λ-calculus we follow the Scott-Strachey approach. Thus variables denote values in a semantic domain, terms of the form $\lambda x.M$ denote continuous functions from the semantic domain to itself and terms of the form MN denote functional application.

2.1 Semantics

The semantics of the λ-calculus adopted here is presented in [Sto77]. Our domains are complete partial orders (CPO). The semantic domain is defined by the following equations:

$$\begin{aligned} V &= B \oplus F \\ F &= V \to V \end{aligned}$$

where V is the domain of values, B is a given domain of basic values and F the domain of functions. $\oplus$ is the coalesced sum. The two injection functions are called in_B and in_F, respectively. We also define the domain of environments, Env by:

$$Env = Var \to V$$

Given an environment ρ, a value v and a variable x we define the environment $\rho[v/x]$ by:

$$\rho[v/x](y) = \begin{cases} v & \text{if } x = y \\ \rho(y) & \text{otherwise} \end{cases}$$

We now define the semantic function:

$$E : \Lambda \to Env \to V$$

as:

$$\begin{array}{lcl} E\|x\|\rho & = & \rho(x) \\ E\|e_1 e_2\|\rho & = & ((E\|e_1\|\rho)_{|F})(E\|e_2\|\rho) \\ E\|\lambda x.E\|\rho & = & in_F(\lambda.vE\|E\|\rho[v/x]) \end{array}$$

2.2 Types

The semantic domain of types is the set of all ideals of V, $\overline{V}$. We assume we are given a map ε from primitive types into $\overline{V}$.

The semantic function $T : Types \to \overline{V}$ is defined as:

$$\begin{array}{lcl} T\|\sigma\|\varepsilon & = & \varepsilon(\sigma) \\ T\|\tau_1 \to \tau_2\|\varepsilon & = & in_F\{f \in V \to V : f(T\|\tau_1\|) \subseteq T\|\tau_2\|\} \end{array}$$

A set of pairs $x : \tau$, where X is a variable and τ is a type is called a basis. We now are in conditions to define a semantic relation between terms and types. Given a basis A and a map ε from primitive types into $\overline{V}$ we define a relation between environments and basis as:

$$\rho :_\varepsilon A \Leftrightarrow \forall_{x:\tau \in A} . \rho(x) \in T\|\tau\|\varepsilon$$

We then define the semantic typing relation $A \models e : \tau$. This relation means that if the basis A holds in the environment then e yields a value of type τ.

Definition 1 *Given a basis A, a λ-term M and a type τ:*

$$A \models M : \tau \Leftrightarrow \forall_{\rho \in Env} . \rho : A \Rightarrow E\|M\|\rho \in T\|\tau\|$$

The first descriptive type system was defined by Haskell Curry in 1934. This system originated the simple type system for the λ-calculus and is defined by the following three rules:

$$AXIOM \qquad A \vdash x : \tau \text{ ,if } x : \tau \in A$$

$$\rightarrow -ELIM. \qquad \frac{A \vdash M : \sigma \rightarrow \tau, A \vdash N : \sigma}{A \vdash MN : \tau}$$

$$\rightarrow -INTRO. \qquad \frac{A \cup \{x : \sigma\} \vdash M : \tau}{A \vdash \lambda x.M : \sigma \rightarrow \tau}$$

This system is the basis of all the descriptive type systems. It has good properties like decidability of type inference and the principal type property.

3 Constrained Type Systems

Constrained type systems have been used for some time to model subtyping. The general idea of these systems is the generalization of the equality constraints implicit in the simple type systems to inequalities which denote type containment.

We define constraints on the domain of ideals of the semantic domain for the λ-calculus. Constraints are just ideals inclusions. Sintactically we represent constraints on types by inequalities of the form $\sigma_1 \sqsubseteq \sigma_2$, where σ_1 and σ_2 are types.

The relation between terms and types is defined by the relation $A * R \vdash M : \tau$ where A is a basis, M a λ-term, τ a type and R a set of constraints, $R = \{\tau_1 \sqsubseteq \sigma_1 \wedge ... \wedge \tau_n \sqsubseteq \sigma_n\}$.

$$AXIOM \qquad A * R \cup \{\sigma \sqsubseteq \tau\} \vdash x : \tau \text{ ,if } x : \sigma \in A$$

$$\rightarrow ELIM \qquad \frac{A * R_1 \vdash M : \alpha' \rightarrow \beta',\ A * R_2 \vdash N : \alpha}{A * R_1 \cup R_2 \cup \{\alpha \sqsubseteq \alpha', \beta' \sqsubseteq \beta\} \vdash MN : \beta}$$

$$\rightarrow INTRO \qquad \frac{A \cup \{x : \alpha'\} * R \vdash M : \beta'}{A * R \cup \{\alpha \sqsubseteq \alpha', \beta' \sqsubseteq \beta\} \vdash \lambda x.M : \alpha \rightarrow \beta}$$

3.1 Semantics

The semantic relation between terms and types is now determined by a set of constraints.

Let as first define what we mean by the evaluation of a set of constraints with respect to a given attribution of ideals to type variables.

Definition 2 *Given a type environment ε and a set of constraints R,*

$$R_\varepsilon \Leftrightarrow \forall_{\tau_1 \sqsubseteq \tau_2 \in R}[T\|\tau_1\|\varepsilon \subseteq T\|\tau_2\|\varepsilon]$$

We can now define the semantic relation $\models$ in the context of constrained type systems:

Definition 3 *Given a basis A, a set of constraints R, a λ-term M and a type τ, let ρ be an environment and ϵ a map between type variables and its semantic value, then:*

$$A * R \models M : \tau \Leftrightarrow \forall_\varepsilon \forall_{\rho \in Env}[\rho :_\varepsilon A \wedge R_\varepsilon \Rightarrow E\|M\|\rho \in T\|\tau\|\varepsilon]$$

We can show that the previous system is sound with respect to the semantic relation between terms, types and sets of constraints.

Theorem 1 *Given a basis A, a set of constraints R, a λ-term M and a type τ:*

$$A * R \vdash M : \tau \Rightarrow A * R \models M : \tau$$

3.2 Constraint Solving

Given a basis A and a term M, a type τ of M, is associated with a set of constraints R of the form $\tau_1 \sqsubseteq \tau_2$. Solving a set of constraints of this form is a semi-unification problem, which is not decidable, [KTU90]. Nevertheless there are subsets of the general set of constraints associated with a typing process which can be solved. As we will see a type system can be viewed as a way of identifing subsets of solutions of the set of constraints, R, associated with a type derivation for a term.

It is easy to see that the subset U of R which can be solved using the unification algorithm, [Rob65], i.e, the solutions corresponding to equations instead of inequalities are the types derived in the Curry type system. If we have $A * R \vdash M : \tau$, where each element of R is of the form $\tau_1 \sqsubseteq \tau_2$, let us find out solutions of the set R' where for each $\tau_1 \sqsubseteq \tau_2 \in R$ we have $\tau_1 = \tau_2 \in R'$. We apply the Robinson unification algorithm which given R' returns a substitution u which when applied to τ and R returns $A * u(R) \vdash u(\tau)$. In $u(R)$ the constraints solved by unification become $u(\tau_1) = u(\tau_2)$ and the others remain in the form of inequalities. The solved constraints can be removed from $u(R)$ originating a set of strict inequalities R_1. Finally we stay with $A * R_1 \vdash u(\tau)$ where R_1 is a set of constraints of the form $\tau_1 \sqsubset \tau_2$.

We now give the following definition:

Definition 4 *A type τ is proper with respect to a basis A and a term M if and only if we have $A \vdash M : \tau$, using unification as a solving strategy for the initial set of constraints.*

If we define a new type system where constraints are equations instead of inequalities we have a system which derives exactly the proper types. It is easy to check that this type system is the Curry type system.

4 Parametric Polymorphism

4.1 The Damas-Milner Type System

The Damas-Milner type system, [DM82], encodes the polymorphic features of the language ML. This type system was investigated for the λ-calculus extended with the *let*-bound variables.

The type language extends the previous one by introducing universally quantified type variables. Type expressions are formed according to the following grammar:

$$\begin{array}{rcl} \tau & ::= & \alpha | \tau' \rightarrow \tau'' \\ \sigma & ::= & \tau | \forall \alpha . \sigma' \end{array}$$

where α ranges over an infinite set of type variables. Type expressions derived from σ are type schemes. Variables bound by $\forall$ are called generic variables. The Damas-Milner type system is defined as: Let x range over variables, α over type variables, e and e' over expressions, τ and τ' over types and σ and σ' over type schemes. The type inference system is given by the following rules:

$$AXIOM \quad A \cup \{x : \sigma\} \vdash x : \sigma$$

$$GEN \qquad \frac{A \vdash e : \sigma \ , \alpha \text{ is not free in } A}{A \vdash e : \forall \alpha . \sigma}$$

$$INST \quad \frac{A \vdash e : \forall\alpha.\sigma}{A \vdash e : \sigma[\tau/\alpha]}$$

$$\rightarrow ELIM \quad \frac{A \vdash e : \tau' \rightarrow \tau, A \vdash e' : \tau'}{A \vdash (ee') : \tau}$$

$$\rightarrow INTRO \quad \frac{A \cup \{x : \tau'\} \vdash e : \tau'}{A \vdash \lambda x.e : \tau' \rightarrow \tau}$$

$$LET \quad \frac{A \vdash e : \sigma, A \cup \{x : \sigma\} \vdash e' : \sigma'}{A \vdash \text{let } x = e \text{ in } e' : \sigma'}$$

4.2 Polymorphic Constrained Types

A polymorphic type, or type scheme, represents the set of types of a term. Thus when adding this notion to the simple type system we have to add to the type language new syntactic elements, the type schemes, which denote sets of types. Thus we have to add to the type system a generalization rule which, given n types $\tau_1, ..., \tau_n$ such that $A \vdash e : \tau_i$, $1 \leq i \leq n$ we can derive $\sigma = \{\tau_1, ..., \tau_n\}$,i.e., $A \vdash e : \sigma$. We also need the opposite: if $A \vdash e : \sigma$ and $\sigma = \{\tau_1, ..., \tau_n\}$ then $A \vdash e : \tau_i$, $1 \leq i \leq n$.

The semantics of a polymorphic type σ is the intersection of its instances semantics. This is why if a value belongs to $T\|\sigma\|$ and $\sigma = \{\tau_1, ..., \tau_n\}$ that value belongs to $T\|\tau_i\|, 1 \leq i \leq n$. Thus in a polymorphic constrained type system we have $\sigma = \{\tau_1, ..., \tau_n\} \Rightarrow \sigma \sqsubseteq \tau_i, 1 \leq i \leq n$. This makes impossible to use unification as the only strategy for constraint solving in the presence of polymorphism. In fact solutions corresponding to instances of type schemes are given by strict inclusion, thus we have to solve strict inequalities in those systems.

Here we define a constrained type system which models parametric polymorphism. We first study the LET rule. Let $\mathcal{T}$ be a polymorphic type such that if $\rho(x) \in T\|\mathcal{T}\|$ then $\rho(x) \in T\|\tau_1\| \wedge ... \wedge \rho(x) \in T\|\tau_n\|$. Thus $\forall_{1 \leq i \leq n} \mathcal{T} \sqsubseteq \tau_i$. The constrained LET rule is:

$$LET \quad \frac{A * R_1 \vdash e : \mathcal{T}_1, A * R_2 \cup \{x : \mathcal{T}_2\} \vdash e' : \tau}{A * R_1 \cup R_2 \cup \{\mathcal{T}_1 \sqsubseteq \mathcal{T}_2, \tau \sqsubseteq \sigma\} \vdash \text{let } x = e \text{ in } e' : \sigma}$$

As we saw before we have to solve strict inequalities when we have polymorphic types. Thus, in polymorphic systems, one has to have an instance rule that derives the instances of a polymorphic type:

$$INST \quad \frac{A \vdash e : \sigma, \sigma \sqsubset \sigma'}{A \vdash e : \sigma'}$$

In the presence of strict inequalities we have to redefine proper types.

Notice that the semantic inclusion relation $\sqsubseteq$ contains another relation $\sqsubseteq^*$ which denotes the instance relation, i.e., $\sigma_1 \sqsubseteq^* \sigma_2$ means that σ_2 is a generic instance of σ_1. It is easy to see that this relation is subsumed by $\sqsubseteq$ according to the semantic definition of a type scheme as the intersection of all its instance semantics.

The definition $\sqsubseteq^*$ follows:

Definition 5 *Given a polymorphic type σ and the set T_σ of instances of σ the relation $\sqsubseteq^*$ is defined as: $\sigma \sqsubseteq^* \tau$ if and only if $\tau \in T_\sigma$.*

We can now define the proper class of polymorphic types:

Definition 6 *Proper types are defined by a constrained type system where we replace $\sqsubseteq$ by $=$, when applied to monomorphic types , and by $\sqsubseteq^*$ when applied to polymorphic types.*

In a polymorphic type system if we derive a set of types for a term and those types have a common representation as a type scheme, we have to derive that type scheme. Otherwise we would have a type scheme denoting a set of derivable types which was not derivable in the system, what goes against the purpose of a polymorphic type system. That principle is represented by the following rule:

$$POLIM \quad \frac{T = \{\tau_i | A * R_i \vdash e : \tau_i\}, \exists_\sigma . T_\sigma \subseteq T}{A * \bigcup_i R_i \vdash e : \sigma}$$

In this rule we need the condition $T_\sigma \subseteq T$, because otherwise we could have a polymorphic type for a term which does not denote properties of this term.

We can now define a polymorphic constrained type system:

$$AXIOM \quad A * \{\sigma' \sqsubseteq \sigma\} \cup R \vdash x : \sigma, \text{ if } x : \sigma' \in A$$

$$\rightarrow ELIM \quad \frac{A * R_1 \vdash M_1 : \alpha' \rightarrow \beta', A * R_2 \vdash N : \alpha}{A * R_1 \cup R_2 \cup \{\alpha \sqsubseteq \alpha', \beta' \sqsubseteq \beta\} \vdash MN : \beta}$$

$$\rightarrow INTRO \quad \frac{A \cup \{x : \alpha'\} * R \vdash M : \beta'}{A * R \cup \{\alpha \sqsubseteq \alpha', \beta' \sqsubseteq \beta\} \vdash \lambda x.M : \alpha \rightarrow \beta}$$

$$LET \quad \frac{A * R_1 \vdash e : \sigma_1, A \cup \{x : \sigma_2\} * R_2 \vdash e' : \tau}{A * R_1 \cup R_2 \cup \{\sigma_1 \sqsubseteq \sigma_2, \tau \sqsubseteq \delta\} \vdash \text{let } x = e \text{ in } e' : \delta}$$

$$INST \quad \frac{A \vdash e : \sigma, \sigma \sqsubseteq \sigma'}{A \vdash e : \sigma'}$$

$$POLIM \quad \frac{T = \{\tau_i | A * R_i \vdash e : \tau_i\}, T_\sigma \subseteq T}{A * \bigcup_i R_i \vdash e : \sigma}$$

The next theorem states the redundancy of the $INST$ rule:

Theorem 2 *Given a basis A, a term M, two types σ and σ' and a set of constraints R such that $A * R \vdash M : \sigma$ and $\sigma \sqsubseteq \sigma'$, we have $A * R \vdash e : \sigma'$*

We prove the soundness of this system:

Theorem 3 *Let A be a basis, M a term , σ a type and R a set of constraints. Then $A * R \vdash M : \sigma \Rightarrow A * R \models M : \sigma$.*

We now define a new type system which derives proper polymorphic types. By the definition of proper polymorphic type we replace $\sqsubseteq$ by $=$ when applied to simple types and by $\sqsubseteq^*$ when applied to type schemes. The proper type system is:

$$AXIOM \qquad A * \{\sigma' \sqsubseteq^* \sigma\} \cup R \vdash x : \sigma, \text{ if } x : \sigma' \in A$$

$$\rightarrow ELIM \qquad \frac{A * R \vdash M_1 : \alpha \rightarrow \beta, A * R \vdash N : \alpha}{A * R \vdash MN : \beta}$$

$$\rightarrow INTRO \qquad \frac{A \cup \{x : \alpha'\} * R \vdash M : \beta}{A * R \vdash \lambda x.M : \alpha \rightarrow \beta}$$

$$LET \qquad \frac{A * R_1 \vdash e : \sigma_1, A \cup \{x : \sigma_2\} * R_2 \vdash e' : \tau}{A * R_1 \cup R_2 \cup \{\sigma_1 \sqsubseteq^* \sigma_2, \tau \sqsubseteq^* \delta\} \vdash \text{let } x = e \text{ in } e' : \delta}$$

$$POLIM \qquad \frac{T = \{\tau_i | A * R_i \vdash e : \tau_i\}, \exists_\sigma . T_\sigma \subseteq T}{A * \bigcup_i R_i \vdash e : \sigma}$$

We prove that the Damas-Milner system derives exactly the proper types for any term. In the proof of our main result there is a fundamental lemma which we refer here:

Theorem 4 *Let $T = \{\tau | A \vdash_{ML} M : \tau\}$. Let π be the principal type of M. Then $\forall_\sigma [T_\sigma \subseteq T_\pi \Rightarrow \pi \sqsubseteq^* \sigma]$.*

The main theorem follows:

Theorem 5 *Let A be a basis, M a term, σ a type and R a set of constraints. $A * R \vdash M : \sigma$ means that σ is a type of M in the proper polymorphic type system and $A \vdash_{ML} M : \sigma$ means that σ is a type of M in the Damas-Milner type system. If we can solve the set of constraints R then :*

$$A \vdash M : \sigma \Leftrightarrow A \vdash_{ML} M : \sigma$$

5 Final Remarks

We used constrained types to model a semantic relation between terms and types stronger that soundness. That semantic relation defines one class of types, the proper types, which type systems should derive to guarantee the descriptive character of the system. The good results obtained for the Curry and the Damas-Milner type systems were already expected because these systems have obvious good properties like decidability of type inference and the principal type property. We are know studing of the application of constrained type systems to the semantic characterization of more problematic systems where type inference is not decidable.

References

[AW93] A. S. Aiken and E. Wimmers. Type inclusion constraints and type inference. *Functional Programming and Programming Architecture*, pages 31–41, June 1993.

[Bar84] H. P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. North-Holland, 1984.

[Chu40] A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 1940.

[Cur34] H.B. Curry. Functionality in combinatory logic. In *Proc. Nat. Acad. Sci. U.S.A.*, volume 20, pages 584–590, 1934.

[DM82] L. Damas and R. Milner. Principal type schemes for functional programs. In *Proc. 9-th ACM Symp. Principles of Programming Languages*, 1982.

[FM88] Y. Fuh and P. Mishra. Type inference with subtypes. In *Proceedings of the 1988 European Symposium on Programming*, 1988.

[KTU90] A. Kfoury, J. Tiuryn, and P. Urzyczyn. The undecidability of the semi-unification problem. *22nd ACM Symposium on Theory of Computation*, 1990.

[PS96] J. Palsberg and S. Smith. Constrained types and their expressiveness. *ACM Transactions on Programming Languages*, 18(5):519–527, 1996.

[Rob65] J. A. Robinson. A machine oriented logic based on the resolution principle. *Journal of ACM*, 12(1):23–41, 1965.

[Sto77] J. Stoy. *Denotational Semantics: The Scott-Strachey approach to Programming Languages.* MIT Press, 1977.

A Appendix

Proof of Theorem 1
We use structural induction on M.

1. Base case: $M \equiv x$ (type variable). In this case $A * R \cup \{\sigma \sqsubseteq \tau\} \vdash x : \tau$, thus $\forall_\varepsilon[\{x : \sigma\} \in A \wedge T\|\sigma\|\varepsilon \subseteq T\|\tau\|\varepsilon]$. By the definition of $\models$ we have $\forall_\rho\forall_\varepsilon[\rho(x) \in T\|\sigma\|\varepsilon \wedge T\|\sigma\|\varepsilon \subseteq T\|\tau\|\varepsilon \Rightarrow \rho(x) \in T\|\tau\|\varepsilon]$, thus $A * R \cup \{\sigma \sqsubseteq \tau\} \models x : \tau$.

2. $M \equiv M_1 M_2$. $A * R_1 \cup R_2 \cup \{\alpha \sqsubseteq \alpha', \beta \sqsubseteq \tau\} \vdash M : \tau$. By the induction hypothesis:

   (a) $A * R_1 \models M_1 : \alpha' \to \beta'$. This is equivalent to

   $$\forall_\rho\forall_\varepsilon[\rho :_\varepsilon A \wedge R_1 \Rightarrow E\|M_1\|\rho \in T\|\alpha' \to \beta'\|\varepsilon]$$

   (b) $A * R_2 \models M_2 : \alpha$, which is equivalent to

   $$\forall_\rho\forall_\varepsilon[\rho :_\varepsilon A \wedge R_2 \Rightarrow E\|M_2\|\rho \in T\|\alpha\|\varepsilon]$$

   If we have as hypothesis $R_3 = \{\alpha \sqsubseteq \alpha', \beta' \sqsubseteq \tau\}$, then we have:

   (a) $A * R_1 \cup R_3 \models M_1 : \alpha' \to \tau$ and

   (b) $A * R_1 \cup R_3 \models M_2 : \alpha'$

   By the definition of $\models$ this implies $A * R_1 \cup R_2 \cup R_3 \models M_1 M_2 : \tau$.

3. $A * R \cup \{\alpha \sqsubseteq \alpha', \beta' \sqsubseteq \beta\} \vdash \lambda x.M : \alpha \to \beta$. We want to show that $\forall_\varepsilon\forall_\rho[\rho :_\varepsilon A \Rightarrow E\|\lambda x.M\|\rho \in T\|\alpha \to \beta\|\varepsilon]$. We have to show that $\forall_\varepsilon\forall_\rho[\rho :_\varepsilon A \Rightarrow E\|M\|\rho[v/x]$ belongs to $T\|\beta\|\varepsilon$ for all v in $T\|\alpha\|\varepsilon]$.

   By the induction hypothesis we know that $A \cup \{x : \alpha'\} * R \models M : \beta'$, which is equivalent to $\forall_\varepsilon\forall_\rho[\rho :_\varepsilon A \cup \{x : \alpha'\} \wedge R \Rightarrow E\|M\|\rho \in T\|\beta'\|\varepsilon]$. Thus for all v in $T\|\alpha\|\varepsilon$, $E\|M\|\rho[v/x]$ belongs to $T\|\beta'\|\varepsilon$. But if we assume that $\alpha \sqsubseteq \alpha'$ the same is true for α, thus for all v belonging to $T\|\alpha\|\varepsilon$, $E\|M\|\rho[v/x]$ belongs to $T\|\beta'\|\varepsilon$. We also have as hypothesis $\beta' \sqsubseteq \beta$, thus the same happens for $T\|\beta\|\varepsilon$. □

The proof of the next theorem is facilitated by the following lemma:

Lemma 1 *$A \cup \{x : \alpha\} * R \vdash M : \beta$ and $\alpha' \sqsubseteq \alpha \Rightarrow A \cup \{x : \alpha'\} * R \vdash M : \beta$*

Proof of Lemma 1

We use structural induction on M. The hypothesis is on a type declaration on the basis A that does not change after a derivation step. We will proof the case where M is a variable x. In this case we have $\{x : \alpha\} * \{\alpha \sqsubseteq \beta\} \vdash x : \beta$. Thus we have $\{x : \alpha'\} * \{\sigma \sqsubseteq \beta\} \vdash x : \beta$ because, by the transitivity of $\sqsubseteq$, $\alpha' \sqsubseteq \alpha \Rightarrow [\alpha \sqsubseteq \beta \Rightarrow \alpha' \sqsubseteq \beta]$. $\square$

Proof of Theorem 2

We will use induction on the lenght of the type derivation tree.

1. Base case: $M \equiv x$. Direct application of $AXIOM$;

2. Terms of the form $\lambda x.M$. In this case we have the following hypothesis:

   (a) $A * R \cup \{\alpha \sqsubseteq \alpha', \beta' \sqsubseteq \beta\} \vdash \lambda x.M : \alpha \to \beta$;

   (b) $A \cup \{x : \alpha'\} * R \vdash M : \beta'$;

   (c) $A \cup \{x : \alpha'\} * R \vdash M : \tau, \forall_\tau.\beta' \sqsubseteq \tau$, by the induction hypothesis.

   By hypothesis we have $\{\beta' \sqsubseteq \beta\}$, thus, $\forall_{\beta^*}.\beta \sqsubseteq \beta^*$ we have $\beta' \sqsubseteq \beta^*$, because $\beta' \sqsubseteq \beta$. Thus $A \cup \{x : \alpha'\} \vdash M : \beta^*$ by c). $\alpha^* \sqsubseteq \alpha \Rightarrow \alpha^* \sqsubseteq \alpha'$ because by hypothesis, $\alpha \sqsubseteq \alpha'$. Thus by lemma 1, $\forall_{\alpha^* \sqsubseteq \alpha}[A \cup \{x : \alpha^*\} \vdash M : \beta^* \Rightarrow A \vdash \lambda x.M : \alpha^* \to \beta^*, \forall_{\alpha \to \beta \sqsubseteq \alpha^* \to \beta^*}$

3. $M \equiv M_1 M_2$. In this case $A \vdash M_1 M_2 : \beta$. We want to show $A \vdash M_1 M_2 : \beta\ \forall_{\beta \sqsubseteq \beta'}$. By hypothesis we know that $A \vdash M_1 : \alpha' \to \beta^*$ if $\beta^* \sqsubseteq \beta$. By the induction hypothesis we also have $A \vdash M_1 : \alpha \to \delta, \forall_{\beta^* \sqsubseteq \delta}$. Any β' such that $\beta \sqsubseteq \beta'$, is such that $\beta^* \sqsubseteq \beta'$, because $\beta^* \sqsubseteq \beta$. Thus $A \vdash M_1 : \alpha' \to \beta'$. By the $\to INTRO$ rule $A \vdash M_1 M_2 : \beta'$.

4. $M \equiv let\ x = M_1\, in\ M_2$. In this case $A \vdash let\ x = M_1\, in\ M_2 : \delta \wedge \tau \sqsubseteq \delta$. By the induction hypothesis $A \cup \{x : \sigma\} \vdash M_2 : \tau' \forall_{\tau \sqsubseteq \tau'}$. For any δ' such that $\delta \sqsubseteq \delta'$ we have $\tau \sqsubseteq \delta'$ (because $\tau \sqsubseteq \delta \wedge \delta \sqsubseteq \delta'$). By the induction hypothesis and the LET rule we conclude $A \vdash let\ x = M_1\, in\ M_2 : \delta'$ for any δ' such that $\delta \sqsubseteq \delta'$.

5. If we apply the $POLIM$ rule, we have $A \vdash e : \sigma$, where σ is a polymorphic type. σ denotes the set of types T_σ, thus any type τ such that $\sigma \sqsubseteq \tau$ belongs to T_σ, and in this case, as $T_\sigma \in T$ where T is the set of all types derivable for e, we have $A \vdash e : \tau$. $\square$

Proof of Theorem 3

The rules $AXIOM$, $\to ELIM$ and $\to INTRO$ are the same as for the simple constrained type system. We show the soundness of rules POLIM and LET. We use induction on the lenght of the derivation tree. The base case is the $AXIOM$ rule which was already proved in theorem 1

1. $POLIM$: In this case we have $A * \bigcup R_i \vdash e : \sigma$. By the previous node in the derivation tree we have $\forall_{\tau \in T_\sigma}[A * R_i \vdash e : \tau]$. By the induction hypothesis $\forall_{\tau \in T_\sigma}[A * R_i \models e : \tau]$, which is equivalent to $\forall_{\tau \in T_\tau} \forall_\rho \forall_\varepsilon [\rho :_\varepsilon A \wedge R_i \Rightarrow E\|e\|\rho \in T\|\tau\|\varepsilon]$ i.e. $\rho : A$ and R_i imply that $E\|e\|\rho$ belongs to $T\|\tau\|\varepsilon, \forall_{\tau \in S_\sigma}$. By the equality $T\|\sigma\|\varepsilon = \bigcap_{\tau \in T_\sigma} T\|\tau\|\varepsilon$ we have $\rho : A \wedge \bigcup R_i \Rightarrow E\|e\|\rho \in T\|\sigma\|\varepsilon$. Note that it is crucial to have $\forall_{\tau \in T_\sigma}[A * R_i \vdash e : \tau]$ because it makes it possible to apply the induction hypothesis and conclude $\forall_{\tau \in T_\sigma}[A * \bigcup R_i \models e : \tau]$.

2. LET: In this case $A * R_1 \cup R_2 \cup \{\sigma \sqsubseteq \sigma', \tau \sqsubseteq \delta\} \vdash let\ x = M_1\, in\ M_2 : \delta$. By the induction hypothesis $A * R_1 \models M_1 : \sigma \wedge A \cup \{x : \sigma'\} * R_2 \models M_2 : \tau$. Assume that $\sigma \sqsubseteq \sigma'$ and $\tau \sqsubseteq \delta$. Then we have:

$$\forall_\rho \forall_\varepsilon [\rho :_\varepsilon A \wedge R_1 \quad \Rightarrow \quad E\|M_1\|\rho \in T\|\sigma\|\varepsilon] \tag{1}$$

$$\forall_\rho \forall_\varepsilon [\rho :_\varepsilon A \wedge \rho(x) \in T\|\sigma'\| \wedge R_2 \quad \Rightarrow \quad E\|M_2\|\rho \in T\|\tau\|\varepsilon] \tag{2}$$

   By theorem 2 if $\rho(x) \in T\|\sigma'\|$ then $E\|M_2\|\rho \in T\|\tau\|\varepsilon$. $E\|\lambda x.M_2\|\rho$ is a function that, when applied to members of $T\|\sigma'\|\varepsilon$, returns members of $T\|\tau\|\varepsilon$. By $\forall_\varepsilon[T\|\sigma\|\varepsilon \subseteq T\|\sigma'\|\varepsilon]$, if one applies $E\|\lambda x.M_2\|\rho$ to a member of $T\|\sigma\|\varepsilon$ one gets a member of $T\|\tau\|\varepsilon$. By $E\|M_1\|\rho \in T\|\sigma\|\varepsilon$ we have $E\|(\lambda x.M_2)M_1\|\rho \in T\|\tau\|\varepsilon$. But $T\|\tau\|\varepsilon \subseteq T\|\delta\|\varepsilon$, thus $E\|(\lambda x.M_2)M_1\|\rho \in T\|\delta\|\varepsilon$. By $E\|let\ x = M_1\, in\ M_2\|\rho = E\|(\lambda x.M_2)M_1\|\rho$ we have $\forall_\rho \forall_\varepsilon[\rho :_\varepsilon A \wedge R_1 \wedge R_2 \wedge \{\sigma \sqsubseteq \sigma', \tau \sqsubseteq \delta\} \Rightarrow E\|let\ x = M_1\, in\ M_2\|\rho \in T\|\delta\|\varepsilon]$, which is equivalent to $A * R_1 \cup R_2 \cup \{\sigma \sqsubseteq \sigma', \tau \sqsubseteq \delta\} \models let\ x = M_1\, in\ M_2 : \delta$ □

Proof of Theorem 4

Suppose there is a $\pi \not\sqsubseteq^* \sigma$ in the conditions stated in theorem 4. In this case σ is not a generic instance of π. Then, there is an instance of σ, τ, such that $\tau \in T_\sigma$ and $\tau \notin T_\pi$. But that goes against the hypothesis $T_\sigma \subseteq T_\pi$. Thus there is not a $\pi \not\sqsubseteq^* \sigma$ in the conditions stated in the theorem. □

Proof of Theorem 5

We will split the proof in two parts; one which proves that $A \vdash_{ML} e : \sigma \Rightarrow A \vdash e : \sigma$ and the other one proves $A \vdash e : \sigma \Rightarrow A \vdash_{ML} e : \sigma$. In both cases we will use induction on the lenght of the type derivation tree.

Case 1: $A \vdash_{ML} e : \sigma \Rightarrow A \vdash e : \sigma$:

1. Base case: we apply the rule $AXIOM_{ML}$. $A \vdash_{ML} x : \sigma$ se $\{x : \sigma\} \in A$ thus $A \vdash e : \sigma$;
2. The cases $\rightarrow ELIM_{ML}$ e $\rightarrow INTRO_{ML}$ are trivial by looking at both type systems;
3. In the case LET_{ML} just consider the solutions corresponding to equalities in the proper type system;
4. Suppose that the last rule used is the $INST$ rule. By the induction hypothesis $A \vdash e : \sigma$ and we want to show $A \vdash e : \sigma' \forall \sigma' [\sigma \sqsubseteq^* \sigma]$. This follows by theorem 2.
5. Suppose that the last rule used is the GEN rule: by the induction hypothesis $A \vdash e : \tau$ $\forall_{\tau \in C} \wedge C \subseteq T_\sigma$. There is a σ such that $A \vdash_{ML} e : \sigma$ and $A \vdash_{ML} e : \tau' \forall_{\tau'} [\sigma \sqsubseteq^* \tau']$, by the rule $INST$. By the previous cases $A \vdash_{ML} e : \tau'$ and τ' is a monomorphic type implies $A \vdash e : \tau'$. Thus $A \vdash e : \tau' \forall_{\tau'} \in T_\sigma$ and $T_\sigma \subseteq T$, with $T = \{\tau_i | A \vdash e : \tau_i\}$. By the rule POLIM we have $A \vdash e : \sigma$.

Case 2: $A \vdash e : \sigma \Rightarrow A \vdash_{ML} e : \sigma$:

1. Case base $(AXIOM)$: $A \vdash x : \sigma \Rightarrow x : \sigma \in A \wedge \sigma \sqsubseteq^* \sigma'$. By the rule $AXIOM_{ML}$ and the rule $INST$, we have $A \vdash_{ML} x : \sigma'$;
2. If the last rule applied is $\rightarrow ELIM$ or $\rightarrow INTRO$ the result follows trivially by looking at both type systems;
3. LET: $A \vdash let\ x = M_1\, in\ M_2 : \tau$. By the induction hypothesis $A \vdash_{ML} M_1 : \sigma \wedge A \cup \{x : \sigma'\} \vdash_{ML} M_2 : \tau \wedge \sigma \sqsubseteq^* \sigma'$. By the $INST$ rule, we have $A \vdash M_1 : \sigma'$ and by the LET_{ML} rule we have $A \vdash_{ML} let\ x = M_1\, in\ M_2 : \tau$;
4. $POLIM$: By the property of the principal type of the Damas-Milner type system we have $T_\pi = \{\tau | A \vdash_{ML} e : \tau\} \Rightarrow A \vdash_{ML} e : \pi$. By the definition of the POLIM rule $A \vdash e : \sigma \Rightarrow A \vdash e : \tau \forall_{\tau \in T_\sigma}$ and $T_\sigma \subseteq T_\pi$. By the induction hypothesis $A \vdash_{ML} e : \tau \forall_{\tau \in T_\sigma}$ and $T_\sigma \subseteq T_\pi$. By theorem 4 $\pi \sqsubseteq^* \sigma$, thus by the rule $INST_{ML}$, $A \vdash_{ML} e : \sigma$ □