A Semi-Syntactic Soundness Proof for HM(X)

François Pottier

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

A Semi-Syntactic Soundness Proof for HM(X)

François Pottier

N° 4150

Mars 2001

THÈME 2

ISSN 0249-6399 ISRN INRIA/RR--4150--FR+ENG

A Semi-Syntactic Soundness Proof for HM(X)

François Pottier

Thème 2 — Génie logiciel
et calcul symbolique
Projet Cristal

Rapport de recherche n° 4150 — Mars 2001 — 13 pages

Abstract: This document gives a soundness proof for the generic constraint-based type inference framework HM(X). Our proof is semi-syntactic. It consists in two steps. The first step is to define a ground type system, where polymorphism is extensional, and prove its correctness in a syntactic way. The second step is to interpret HM(X) judgements as (sets of) judgements in the underlying system, which gives a logical view of polymorphism and constraints. Overall, the approach may be seen as more modular than a purely syntactic approach: because polymorphism and constraints are dealt with separately, they do not clutter the subject reduction proof. However, it yields a slightly weaker result: it only establishes type soundness, rather than subject reduction, for HM(X).

Key-words: constraint-based type inference, type soundness

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau, Rocquencourt, BP 105, 78153 Le Chesnay Cedex (France)
Téléphone : +33 1 39 63 55 11 — Télécopie : +33 1 39 63 53 30

Une preuve de correction semi-syntaxique pour HM(X)

Résumé : Ce document contient une preuve de correction du système générique d'inférence de types à base de contraintes HM(X). Notre preuve est semi-syntaxique. Elle est constituée de deux étapes. La première consiste à définir un système de types sans variables, doté d'une notion extensionnelle de polymorphisme, et d'en prouver la correction de façon syntaxique. La seconde étape est d'interpréter les jugements de HM(X) en tant qu'ensembles de jugements dans le système sous-jacent, ce qui permet une vision logique du polymorphisme et des contraintes. Au total, l'approche peut être considérée comme plus modulaire qu'une approche purement syntaxique: parce que le polymorphisme et les contraintes sont traités séparément, ils n'obscurcissent pas la preuve d'auto-réduction. Cependant, elle conduit à un résultat légèrement plus faible: elle n'établit que la correction du typage, et non l'auto-réduction, pour HM(X).

Mots-clés : inférence de types à base de contraintes, correction du typage

1 Introduction

Type soundness proofs for programming languages usually fall in one of two categories: denotational or syntactic. Approaches based on denotational semantics view types as (particular) sets of values [3, 1]. Type soundness then states that $e : \tau$ implies $[\![e]\!] \in [\![\tau]\!]$, i.e. the meaning of every expression is a member of the meaning of its type. This is established by structural induction on the derivation of $e : \tau$. On the other hand, approaches based on operational semantics usually offer a purely syntactic view of types, which are not given any logical interpretation [10]. Rather, type soundness is proved by means of two complementary results, *subject reduction*, which states that reduction preserves types, and *progress*, which states that no well-typed expression is stuck (i.e. constitutes a runtime error). Subject reduction is established by considering every form of redex and examining all of its possible type derivations. The two approaches have distinct advantages:

- A proof by induction on typing derivations is, in our experience, more elementary and more robust than a subject reduction proof. Indeed, if the operational semantics is complex, a single redex may involve many language constructs at once. Then, examining the type derivation of the redex and building a type derivation for its reduct can be a difficult task. Thus, subject reduction proofs tend to have a small number of heavy cases, while denotational soundness proofs exhibit a greater number of smaller cases. In particular, in the syntactic approach, non-syntax-directed typing rules require (sometimes tricky) normalization lemmas, while, in the denotational framework, they simply add extra proof cases.

- On the other hand, an operational semantics is often significantly easier to define and to comprehend than a denotational one. For this reason, many programming languages do not have a denotational semantics.

In this report, we suggest a third approach, which attempts to strike a balance between the two, by starting with an operational semantics, but still writing part of the proof in a logical style, i.e. by induction on typing derivations. We illustrate our approach by giving a type soundness proof for HM(X), a generic constraint-based type system originally defined by Odersky, Sulzmann and Wehr [5] for the λ-calculus with let, which we extend here with imperative features.

We begin by giving a type system, called B(T), which enjoys subject reduction and progress properties. It is a *ground* type system in the sense that its monotypes and polytypes are atoms taken from mathematical sets; in other words, they are not terms, and there is no notion of type variable. Because of this feature, the system is very close to the simply-typed λ-calculus, even though it does have rank-1 polymorphism. As a result, the subject reduction proof is straightforward – it is almost exactly the same as if polymorphism was absent.

Then, we define the type system we are truly interested in. It is HM(X) with a few technical enhancements. We show that its judgements can be interpreted as sets of B(T)

judgements. Type soundness then follows easily. The proof is by induction on type derivations. Thus, as mentioned above, each case requires only elementary reasoning, and non-syntax-directed rules (of which there are 5) do not require separate normalization lemmas.

Overall, the approach has the advantage of enabling a subject reduction proof where constraints are absent and polymorphism is essentially invisible, thus avoiding clutter. Constraints and polymorphism are then dealt with in a second, orthogonal step. When the operational semantics is simple, as in this report, the advantage over a direct syntactic proof may not be obvious. However, we believe it can become significant when dealing with more complex operational semantics. This is illustrated by joint papers with Sylvain Conchon and Vincent Simonet, where we apply the semi-syntactic approach to a constraint-based type system for the join-calculus [2] and to an information-flow-aware, constraint-based type system for ML [6], respectively.

Although this report is self-contained, some familiarity with HM(X) [5, 8, 7] will be helpful.

2 The language

The programming language we are interested in is core ML, including references, equipped with a small-step call-by-value operational semantics.

Following Wright [9], we restrict polymorphism to values, so as to avoid adverse interaction with the language's imperative features. This simply consists in removing the production $E ::= \text{let } x = E \text{ in } e$ in the definition of evaluation contexts, and, accordingly, using $e ::= \text{let } x = v \text{ in } e$ as the only legal form of let definition.

Let $x, y, \ldots$ (resp. $l, m, \ldots$) range over a denumerable set of identifiers (resp. memory locations). Values, expressions and evaluation contexts are defined as follows:

$$\begin{aligned} v &::= l \mid \lambda x.e \mid \text{ref} \mid := \mid (:= l) \mid ! \\ e &::= x \mid v \mid e\, e \mid \text{let } x = v \text{ in } e \\ E &::= [] \mid E\, e \mid v\, E \end{aligned}$$

An expression is said to be closed iff it has no free identifiers. A store σ is a partial map from locations to values. We write $\varnothing$ for the empty store. If $l \notin \text{dom}(\sigma)$, then $\sigma \oplus (l \mapsto v)$ denotes the store which extends σ and maps l to v. If $l \in \text{dom}(\sigma)$, then $\sigma \leftarrow (l \mapsto v)$ denotes the store which maps l to v and agrees with σ otherwise. A configuration e/σ is a pair of an expression e and a store σ. Then, the semantics is given by

$$\begin{array}{rcll} (\lambda x.e)\, v/\sigma & \to & e[v/x]/\sigma & (\beta) \\ \text{let } x = v \text{ in } e/\sigma & \to & e[v/x]/\sigma & (let) \\ \text{ref}\, v/\sigma & \to & l/\sigma \oplus (l \mapsto v) & (ref) \\ :=\, l\, v/\sigma & \to & v/\sigma \leftarrow (l \mapsto v) & (assign) \\ !\, l/\sigma & \to & \sigma(l)/\sigma & (deref) \\ E[e]/\sigma & \to & E[e']/\sigma' \quad \text{when } e/\sigma \to e'/\sigma' & (context) \end{array}$$

3 The system B(T)

3.1 Assumptions

The type system B(T) is parameterized by a universe T of so-called *monotypes*, also known as "ground types" in the literature.

Assumptions. *Let $(T, \leq)$ be a partially ordered set. Its elements, denoted by t, are called* monotypes. *Let $\rightarrow$ be a total function from $T \times T$ into T, such that $t_0 \rightarrow t_1 \leq t'_0 \rightarrow t'_1$ implies $t'_0 \leq t_0$ and $t_1 \leq t'_1$. Let* ref *be a total function from T to T, such that $t\,\mathrm{ref} \leq t'\,\mathrm{ref}$ implies $t = t'$. We require $t_0 \rightarrow t_1 \leq t\,\mathrm{ref}$ and $t\,\mathrm{ref} \leq t_0 \rightarrow t_1$ to be false for any $t, t_0, t_1 \in T$.*

3.2 Definition

Under these assumptions, we define a simple type system, called B(T), where B stands for "basic".

Definition 1 *If V is a subset of T, the* cone *generated by V within T, denoted by $\uparrow V$, is $\{t \in T\,;\, \exists v \in V \quad v \leq t\}$. V is said to be* upward-closed *if and only if $V = \uparrow V$.*

Definition 2 *Let S be the set of all non-empty, upward-closed subsets of T. Its elements, denoted by s, are called* polytypes.

Note that $\leq$, $\rightarrow$ and ref operate on T. Furthermore, S is defined on top of T; there is no way to inject S back into T. In other words, this presentation allows *rank-1* polymorphism only; impredicative polymorphism is ruled out. This is in keeping with the Hindley-Milner family of type systems [4, 5].

Definition 3 *A polytype environment is a partial mapping from identifiers into S. Given an environment Γ, an identifier x and a polytype s, let $\Gamma[x \mapsto s]$ stand for the function which maps x to s and agrees with Γ otherwise. We write $\Gamma[x \mapsto t]$ for $\Gamma[x \mapsto \uparrow\{t\}]$.*

Definition 4 *A memory environment is a partial mapping from memory locations into T. Given an environment M, a location l and a monotype t, let $M[l \mapsto t]$ stand for the function which maps l to t and agrees with M otherwise.*

Definition 5 *A* judgement *in the system $B(T)$ is defined as a quadruple of a polytype environment Γ, a memory environment M, an expression e and a monotype t, written $\Gamma, M \vDash e : t$, derivable using the rules given in figure 1. We write $\Gamma, M \vDash e : s$ if and only if, for all t in s, $\Gamma, M \vDash e : t$ holds.*

Definition 6 *A* configuration judgement *in the system $B(T)$ is defined as a triple of a polytype environment Γ, a configuration e/σ and a monotype t, written $\Gamma \vDash e/\sigma : t$, derivable using the rules given in figure 1.*

Expressions

$$\textsc{b-Var}\ \frac{t \in \Gamma(x)}{\Gamma, M \vDash x : t} \qquad \textsc{b-Loc}\ \Gamma, M \vDash l : M(l)\,\mathrm{ref} \qquad \textsc{b-Abs}\ \frac{\Gamma[x \mapsto t], M \vDash e : t'}{\Gamma, M \vDash \lambda x.e : t \to t'} \qquad \textsc{b-Ref}\ \Gamma, M \vDash \mathrm{ref} : t \to t\,\mathrm{ref}$$

$$\textsc{b-Assign}\ \Gamma, M \vDash := : t\,\mathrm{ref} \to t \to t \qquad \textsc{b-Deref}\ \Gamma, M \vDash ! : t\,\mathrm{ref} \to t \qquad \textsc{b-App}\ \frac{\Gamma, M \vDash e_1 : t_2 \to t \quad \Gamma, M \vDash e_2 : t_2}{\Gamma, M \vDash e_1\, e_2 : t}$$

$$\textsc{b-Let}\ \frac{\Gamma, M \vDash v : s \quad \Gamma[x \mapsto s], M \vDash e : t}{\Gamma, M \vDash \mathrm{let}\ x = v\ \mathrm{in}\ e : t} \qquad \textsc{b-Sub}\ \frac{\Gamma, M \vDash e : t \qquad t \leq t'}{\Gamma, M \vDash e : t'}$$

Configurations

$$\textsc{b-Store}\ \frac{\mathrm{dom}(M) = \mathrm{dom}(\sigma) \quad \forall l \in \mathrm{dom}(\sigma) \quad \varnothing, M \vDash \sigma(l) : M(l)}{M \vDash \sigma} \qquad \textsc{b-Conf}\ \frac{\Gamma, M \vDash e : t \qquad M \vDash \sigma}{\Gamma \vDash e/\sigma : t}$$

Figure 1: The system B(T)

B(T) is the simply-typed λ-calculus with subtyping, extended with rank-1 polymorphism. Remarkably, polymorphism is dealt with in an *extensional* way. Indeed, a polytype is nothing but the set of its monotype instances. A value may be given a polytype s if and only if it has every monotype t in s (rule B-Let); conversely, if the binding $x : s$ occurs in the environment, then x is assumed to have every monotype t in s (rule B-Var). It seems difficult to conceive a more straightforward treatment.

3.3 Properties

It is clear that a closed expression e is well-typed in some environment if and only if it is well-typed in every environment. In such a case, we write $M \vDash e : t$ (resp. $\vDash e/\sigma : t$) instead of $\Gamma, M \vDash e : t$ (resp. $\Gamma \vDash e/\sigma : t$) for some (or for all) Γ.

Lemma 1 (Substitution) *If v is closed, then $\Gamma[x \mapsto s], M \vDash e : t$ and $M \vDash v : s$ imply $\Gamma, M \vDash e[v/x] : t$.*

Proof. By induction on the derivation of $\Gamma[x \mapsto s], M \vDash e : t$.

Case B-VAR. If e is x, then $\Gamma[x \mapsto s], M \vDash e : t$ implies $t \in s$. Given $M \vDash v : s$, this yields $M \vDash v : t$, which may be read $\Gamma, M \vDash e[v/x] : t$. If $e \neq x$, then the result stems from $\Gamma[x \mapsto s](e) = \Gamma(e)$ and $e[v/x] = e$.

Case B-ABS. Then, e must be of the form $\lambda y.e'$, and t is of the form $t_0 \to t_1$. The premise is $\Gamma[x \mapsto s][y \mapsto t_0], M \vDash e' : t_1$. It is easy to check that typing judgements are stable under α-conversion. So, w.l.o.g., we will assume $y \neq x$. Then, $\Gamma[x \mapsto s][y \mapsto t_0]$ coincides with $\Gamma[y \mapsto t_0][x \mapsto s]$. We conclude by applying the induction hypothesis followed by an instance of B-ABS.

Case B-LET. Similar.

Cases B-LOC, B-REF, B-ASSIGN, B-DEREF, B-APP, B-SUB. Immediate. □

Lemma 2 (Subject Reduction) *Let $e/\sigma \to e'/\sigma'$, where e, e' are closed. Assume $M \vDash e : t$ and $M \vDash \sigma$. Then, there exists a memory environment M', which extends M, such that $M' \vDash e' : t$ and $M' \vDash \sigma'$.*

Proof. By induction on the derivation of $e \to e'$. We assume, w.l.o.g., that the derivation of $M \vDash e : t$ does not end with an instance of B-SUB.

Case (β). Then, e is of the form $(\lambda x.f)\, v$, while e' is $f[v/x]$. The derivation of $M \vDash e : t$ must end with an instance of rule B-APP, whose premises are $M \vDash \lambda x.f : t_2 \to t$ and $M \vDash v : t_2$, for some $t_2 \in T$. The former's derivation must end with an instance of B-ABS, possibly followed by a number of instances of B-SUB. As a result, there must exist t_2', t' such that $x \mapsto t_2', M \vDash f : t'$ and $t_2' \to t' \leq t_2 \to t$. This yields $t_2 \leq t_2'$ and $t' \leq t$. By B-SUB, the former implies $M \vDash v : t_2'$, and the latter implies $x \mapsto t_2', M \vDash f : t$. Then, lemma 1 yields $M \vDash f[v/x] : t$.

Case (*let*). Then, e is of the form let $x = v$ in f, while e' is $f[v/x]$. The derivation of $M \vDash e : t$ must end with an instance of rule B-LET, whose premises are $M \vDash v : s$ and $x \mapsto s, M \vDash f : t$, for some $s \in S$. Lemma 1 yields $M \vDash f[v/x] : t$.

Case (*ref*). Then, e is of the form $(\mathrm{ref}\, v)$, while e' is l and σ' is $\sigma \oplus (l \mapsto v)$. The derivation of $M \vDash e : t$ must end with an instance of rule B-APP, whose premises are $M \vDash \mathrm{ref} : t' \to t$ and $M \vDash v : t'$, for some $t' \in T$. The former must be a consequence of B-REF and B-SUB, so there must exist some $t'' \in T$ such that $t'' \to t''\,\mathrm{ref} \leq t' \to t$. (This implies $t' \leq t''$ and $t''\,\mathrm{ref} \leq t$.) Define $M' = M[l \mapsto t'']$. Because $M \vDash \sigma$ holds, we have $\mathrm{dom}(M) = \mathrm{dom}(\sigma)$; because $\sigma \oplus (l \mapsto v)$ is defined, $l \notin \mathrm{dom}(\sigma)$ holds. So, M' extends M. Furthermore, by B-LOC, $M' \vDash l : t''\,\mathrm{ref}$ holds; by B-SUB, this yields $M' \vDash l : t$. Lastly, by B-SUB, $M \vDash v : t''$ holds. Because l cannot appear free in v, this implies $M' \vDash v : t''$; similarly, we have $M' \vDash \sigma$. It follows that $M' \vDash \sigma \oplus (l \mapsto v)$.

Case (*assign*). Then, e is of the form $(:= l\, v)$, while e' is v and σ' is $\sigma \leftarrow (l \mapsto v)$. The derivation of $M \vDash e : t$ must end with an instance of rule B-APP, whose premises are $M \vDash (:= l) : t_1 \to t$ and $M \vDash v : t_1$ for some $t_1 \in T$. By B-SUB and B-APP, the former yields $M \vDash := : t_3 \to t_2$ and $M \vDash l : t_3$, for some $t_2, t_3 \in T$ such that $t_2 \leq t_1 \to t$. Lastly, by B-SUB and B-ASSIGN, we deduce that $t_4\,\mathrm{ref} \to t_4 \to t_4 \leq t_3 \to t_2$ holds for some $t_4 \in T$. By decomposing subtyping relationships involving arrow types, we obtain $t_3 \leq t_4\,\mathrm{ref}$ and $t_1 \leq t_4 \leq t$. By B-SUB, these yield $M \vDash l : t_4\,\mathrm{ref}$ and $M \vDash v : t_4$. By B-LOC, B-SUB and

invariance of the ref type constructor, the former implies $M(l) = t_4$. As a result, the latter gives $M \vDash \sigma \leftarrow (l \mapsto v)$. Lastly, by B-SUB, $M \vDash v : t$ holds.

Case (*deref*). Then, e is of the form $(!\,l)$, while e' is $\sigma(l)$. The derivation of $M \vDash e : t$ must end with an instance of rule B-APP, whose premises are $M \vDash (!\,l) : t_1 \to t$ and $M \vDash l : t_1$, for some $t_1 \in T$. By B-DEREF and B-SUB, $t_2\,\mathrm{ref} \to t_2 \leq t_1 \to t$ must hold for some $t_2 \in T$. This implies $t_2 \leq t$ and $t_1 \leq t_2\,\mathrm{ref}$. As a result of the latter, B-SUB yields $M \vDash l : t_2\,\mathrm{ref}$. As above, this implies $M(l) = t_2$. Then, $M \vDash \sigma$ yields $M \vDash \sigma(l) : t_2$. By B-SUB, this entails $M \vDash \sigma(l) : t$.

Case (*context*). Then, e (resp e') is of the form $E[f]$ (resp. $E[f']$), for some evaluation context E and closed expressions f, f' such that $f/\sigma \to f'/\sigma'$. The derivation of $M \vDash e : t$ includes a derivation of $M \vDash f : t'$ for some $t' \in T$, i.e. it is of the form $\mathcal{D}[M \vDash f : t']$, where $\mathcal{D}$ is a derivation with a hole. By induction hypothesis, there exists a memory environment M' which extends M such that $M' \vDash f' : t'$ and $M' \vDash \sigma'$ hold. Because M' extends M, $\mathcal{D}[M'/M][M' \vDash f' : t']$ is a derivation of $M' \vDash e' : t$. □

As a corollary, we obtain

Theorem 1 (Subject Reduction) *If $e/\sigma \to e'/\sigma'$, where e, e' are closed, then $\vDash e/\sigma : t$ implies $\vDash e'/\sigma' : t$.*

Proof. By B-CONF and lemma 2. □

Theorem 2 (Progress) *If a closed irreducible configuration e/σ is well-typed, then e is a value.*

Proof. Imagine e/σ is irreducible, yet e isn't a value. Then, by case analysis, e must be of the form $E[f]$, where E is an evaluation context and one of the following holds:

1. f is of the form $(l\,v)$. Assuming e is well-typed, so is f. By B-APP, we must have $M \vDash l : t \to t'$ for some memory environment M and some $t, t' \in T$. By B-LOC and B-SUB, this requires $M(l)\,\mathrm{ref} \leq t \to t'$, a contradiction.
2. f is of the form $(:= v)$ or $(!\,v)$, where v is not a memory location. Similar, this time using the fact that $t \to t' \leq t''\,\mathrm{ref}$ is a contradiction.
3. f is of the form $(:= l\,v)$ or $(!\,l)$, where $l \notin \mathrm{dom}(\sigma)$. Because e/σ is well-typed, we must have $M \vDash e : t$ and $M \vDash \sigma$ for some memory environment M and some $t \in T$. Because l occurs in e, we must have $l \in \mathrm{dom}(M)$. Yet, $M \vDash \sigma$ requires $\mathrm{dom}(M) = \mathrm{dom}(\sigma)$, a contradiction. □

4 The system HM(X)

From here on, we will only consider *source language* expressions, i.e. expressions which do not contain memory locations. Indeed, the notion of memory location will no longer be useful, because locations appear only during reduction, and we will not state a subject reduction theorem. When e is a source language expression, the B(T) judgement $\Gamma, M \vDash e : t$ will be written $\Gamma \vDash e : t$, to emphasize the fact that M is then irrelevant.

4.1 Assumptions

Like B(T), HM(X) is parameterized by a set of monotypes T. It is further parameterized by a first-order logic X, interpreted in T, whose variables, terms and formulas are respectively called *type variables*, *types* and *constraints*. The logic allows describing subsets of T as constraints. Provided constraint satisfiability is decidable, this gives rise to a type system where type checking is decidable.

Our presentation differs from the original [5, 8, 7] by explicitly viewing constraints as formulas interpreted in T, rather than as elements of an abstract cylindric constraint system. This presentation is more concise, and gives us the ability to explicitly manipulate *solutions* of constraints, an essential requirement in our formulation of type soundness. Even though we lose some generality with respect to the cylindric-system approach, the framework seems to remain general enough for many purposes.

Assumptions. *We assume given* $(T, \leq, \to, \mathrm{ref})$ *as in section 3. Furthermore, we assume given a constraint logic* X *which defines a syntax of* types *and* constraints*:*

$$\tau ::= \alpha, \beta, \ldots \mid \tau \to \tau \mid \tau\,\mathrm{ref} \mid \ldots$$
$$C ::= \mathbf{true} \mid \tau \leq \tau \mid C \wedge C \mid \exists \bar{\alpha}.C \mid \ldots$$

($\alpha, \beta, \gamma, \ldots$ *range over a denumerable set of* type variables $\mathcal{V}$*.)*

Let an assignment ρ *be a total mapping* ρ *from* $\mathcal{V}$ *into* T*. The logic* X *must be equipped with an* interpretation *in* T*, that is, an extension of assignments to arbitrary types, and a constraint satisfaction predicate* $\vdash$*, whose arguments are an assignment and a constraint. The interpretation must be standard, i.e. satisfy the following laws:*

$$\begin{array}{rcl}
\rho(\tau \to \tau') & = & \rho(\tau) \to \rho(\tau') \\
\rho(\tau\,\mathrm{ref}) & = & \rho(\tau)\,\mathrm{ref} \\
\multicolumn{3}{c}{\rho \vdash \mathbf{true}} \\
\rho \vdash \alpha = \beta \to \gamma & \Longleftrightarrow & \rho(\alpha) = \rho(\beta) \to \rho(\gamma) \\
\rho \vdash \alpha_0 \leq \alpha_1 & \Longleftrightarrow & \rho(\alpha_0) \leq \rho(\alpha_1) \\
\rho \vdash C_0 \wedge C_1 & \Longleftrightarrow & (\rho \vdash C_0) \wedge (\rho \vdash C_1) \\
\rho \vdash \exists \bar{\alpha}.C & \Longleftrightarrow & \exists \rho' \quad (\rho \setminus \bar{\alpha} = \rho' \setminus \bar{\alpha}) \wedge \rho' \vdash C
\end{array}$$

($\rho \setminus \bar{\alpha}$ *denotes the restriction of* ρ *to* $\mathcal{V} \setminus \bar{\alpha}$*.) We write* $C \Vdash C'$ *if and only if* C *entails* C'*, i.e. if and only if every solution* ρ *of* C *satisfies* C' *as well.*

The syntax is only partially specified; this allows other forms, not known in this report, to be introduced at a later stage. Their interpretation is unspecified at this point.

4.2 Definition

HM(X) has *constrained type schemes*, where a number of type variables $\bar{\alpha}$ are universally quantified, subject to a constraint C.

Definition 7 *A* type scheme *is a triple of a set of quantifiers $\bar{\alpha}$, a constraint C, and a type τ; we write $\sigma ::= \forall\bar{\alpha}[C].\tau$. The type variables in $\bar{\alpha}$ are bound in σ; type schemes are considered equal modulo α-conversion. By abuse of notation, a type τ may be viewed as a type scheme $\forall\varnothing[\mathbf{true}].\tau$.*

Definition 8 *Given a type scheme $\sigma = \forall\bar{\alpha}[D].\tau$, and a constraint C, we say that σ is* consistent *with respect to C, and we write $C \Vdash \sigma$, if and only if $C \Vdash \exists\bar{\alpha}.D$.*

Definition 9 *An environment Γ is a sequence of bindings of the form $x : \sigma$, where x is an identifier and σ is a type scheme. We let $\Gamma; x : \sigma$ denote the environment obtained by appending the binding $x : \sigma$ to Γ. We let $\Gamma(x)$ denote the type scheme which appears in the rightmost binding of x in Γ, if any such binding exists; $\Gamma(x)$ is undefined otherwise.*

Definition 10 *A* judgement *in the system HM(X) is defined as a quadruple of a* satisfiable *constraint C, an environment Γ, an expression e and a type scheme σ, written $C, \Gamma \vdash e : \sigma$, derivable using the rules given in figure 2.*

$$\textsc{Var}\ \frac{\Gamma(x) = \sigma \qquad C \Vdash \sigma}{C, \Gamma \vdash x : \sigma} \qquad \textsc{Abs}\ \frac{C, \Gamma; x : \tau \vdash e : \tau'}{C, \Gamma \vdash \lambda x.e : \tau \to \tau'} \qquad \textsc{Ref}\ C, \Gamma \vdash \mathrm{ref} : \forall\alpha.\alpha \to \alpha\ \mathrm{ref}$$

$$\textsc{Assign}\ C, \Gamma \vdash := : \forall\alpha.\alpha\ \mathrm{ref} \to \alpha \to \alpha \qquad \textsc{Deref}\ C, \Gamma \vdash\ ! : \forall\alpha.\alpha\ \mathrm{ref} \to \alpha$$

$$\textsc{App}\ \frac{C, \Gamma \vdash e_1 : \tau_2 \to \tau \qquad C, \Gamma \vdash e_2 : \tau_2}{C, \Gamma \vdash e_1\, e_2 : \tau} \qquad \textsc{Let}\ \frac{C, \Gamma \vdash v : \sigma \qquad C, \Gamma; x : \sigma \vdash e : \tau}{C, \Gamma \vdash \mathrm{let}\ x = v\ \mathrm{in}\ e : \tau}$$

$$\textsc{Sub}\ \frac{C, \Gamma \vdash e : \tau \qquad C \Vdash \tau \leq \tau'}{C, \Gamma \vdash e : \tau'} \qquad \textsc{Weaken}\ \frac{C', \Gamma \vdash e : \sigma \qquad C \Vdash C'}{C, \Gamma \vdash e : \sigma}$$

$$\forall\ \textsc{Intro}\ \frac{C \wedge D, \Gamma \vdash v : \tau \qquad \bar{\alpha} \cap \mathrm{fv}(C, \Gamma) = \varnothing}{C \wedge \exists\bar{\alpha}.D, \Gamma \vdash v : \forall\bar{\alpha}[D].\tau} \qquad \forall\ \textsc{Elim}\ \frac{C, \Gamma \vdash v : \forall\bar{\alpha}[D].\tau}{C \wedge D, \Gamma \vdash v : \tau}$$

$$\exists\ \textsc{Intro}\ \frac{C, \Gamma \vdash e : \sigma \qquad \bar{\alpha} \cap \mathrm{fv}(\Gamma, \sigma) = \varnothing}{\exists\bar{\alpha}.C, \Gamma \vdash e : \sigma}$$

Figure 2: The system HM(X)

HM(X) differs from B(T) by replacing monotypes with type variables, polytypes with type schemes, and parameterizing every judgement with a constraint C, which represents an assumption about its free type variables. Rule WEAKEN allows strengthening this assumption, while $\exists$ INTRO allows hiding auxiliary type variables which appear nowhere but in the assumption itself. These rules allow constraint simplification.

Our treatment of constrained polymorphism is standard. Whereas B(T) takes an extensional view of polymorphism, HM(X) offers the usual, intensional view. Type schemes are introduced by rule $\forall$ INTRO, and eliminated by $\forall$ ELIM. Because implicit α-conversion is allowed, $\forall$ ELIM is a non-deterministic rule.

4.3 Properties

This section gives a type soundness proof for HM(X) by showing that it is safe with respect to B(T). That is, we show that every (valid) judgement $C, \Gamma \vdash e : \sigma$ gives rise to a set of (valid) B(T) judgements. Thus, we give logical (rather than syntactic) meaning to HM(X) judgements, yielding a concise and natural proof. As a whole, the approach is still semi-syntactic, because B(T) itself has been proven correct in a syntactic way.

Let us first state a useful invariant about the structure of HM(X) judgements.

Lemma 3 (Consistency) *$C, \Gamma \vdash e : \sigma$ implies $C \Vdash \sigma$.*

Proof. By induction on the type derivation. Whenever σ carries a **true** constraint, the result is immediate. Only three cases then remain to be examined.

Case VAR. Then, the second premise is $C \Vdash \sigma$.

Case $\forall$ INTRO. Then, we must check $C \wedge \exists\bar{\alpha}.D \Vdash \exists\bar{\alpha}.D$, a tautology.

Case $\exists$ INTRO. Then, the induction hypothesis yields $C \Vdash \sigma$. Because $\bar{\alpha} \cap \mathrm{fv}(\sigma) = \varnothing$, this entails $\exists\bar{\alpha}.C \Vdash \sigma$. □

We now present our interpretation of HM(X) judgements as sets of B(T) judgements. We begin by defining how type schemes are mapped to polytypes. (The definition below is valid because it is stable under α-conversion of type schemes.)

Definition 11 *The* interpretation *of a type scheme $\sigma = \forall\bar{\alpha}[D].\tau$ with respect to an assignment ρ is defined by*

$$[\![\sigma]\!]_\rho = \uparrow\{\rho'(\tau)\,;\, (\rho \setminus \bar{\alpha} = \rho' \setminus \bar{\alpha}) \wedge \rho' \vdash D\}$$

if $\rho \vdash \exists\bar{\alpha}.D$ holds; it is undefined otherwise. Notice that, when it is defined, $[\![\sigma]\!]_\rho$ is a polytype, i.e. an element of S. Note also that $[\![\tau]\!]_\rho$ is $\uparrow\{\rho(\tau)\}$.

The interpretation of an environment Γ under an assignment ρ, denoted $[\![\Gamma]\!]_\rho$, is the composition $[\![\cdot]\!]_\rho \circ \Gamma$, where both $[\![\cdot]\!]_\rho$ and Γ are viewed as partial functions. $[\![\Gamma]\!]_\rho$ is a polytype environment in the sense of definition 3.

By lemma 3, whenever $C, \Gamma \vdash e : \sigma$ and $\rho \vdash C$ hold, $[\![\sigma]\!]_\rho$ is defined. This allows us to state:

Theorem 3 (Interpretation) *$C, \Gamma \vdash e : \sigma$ and $\rho \vdash C$ imply $[\![\Gamma]\!]_\rho \vDash e : [\![\sigma]\!]_\rho$.*

Proof. By structural induction on the derivation of the input judgement. We use exactly the notations of figure 2. In each case, let ρ be some solution of the constraint which appears in the judgement's conclusion; let t be some element of the interpretation (under ρ) of the type scheme which appears in the judgement's conclusion.

Case VAR. Then, $t \in [\![\sigma]\!]_\rho = [\![\Gamma(x)]\!]_\rho = [\![\Gamma]\!]_\rho(x)$. By B-VAR, $[\![\Gamma]\!]_\rho \vDash x : t$ holds.

Case ABS. The induction hypothesis, specialized at $\rho(\tau')$, yields $[\![\Gamma]\!]_\rho[x \mapsto [\![\tau]\!]_\rho] \vDash e : \rho(\tau')$. By B-ABS, this entails $[\![\Gamma]\!]_\rho \vDash \lambda x.e : \rho(\tau) \to \rho(\tau')$, i.e. $[\![\Gamma]\!]_\rho \vDash \lambda x.e : \rho(\tau \to \tau')$. Recalling $t \in [\![\tau \to \tau']\!]_\rho$, B-SUB yields $[\![\Gamma]\!]_\rho \vDash \lambda x.e : t$.

Case REF. The hypothesis $t \in [\![\forall\alpha.\alpha \to \alpha\,\mathrm{ref}]\!]_\rho$ yields $t' \to t'\,\mathrm{ref} \leq t$ for some $t' \in T$. The result follows by B-REF and B-SUB.

Cases ASSIGN, DEREF. Similar.

Case APP. The induction hypotheses, suitably specialized, yield $[\![\Gamma]\!]_\rho \vDash e_1 : \rho(\tau_2 \to \tau)$ and $[\![\Gamma]\!]_\rho \vDash e_2 : \rho(\tau_2)$. By B-APP, $[\![\Gamma]\!]_\rho \vDash e_1\,e_2 : \rho(\tau)$ holds. Recalling $t \in [\![\tau]\!]_\rho$, B-SUB yields $[\![\Gamma]\!]_\rho \vDash e_1\,e_2 : t$.

Case LET. Applying the induction hypothesis to the first premise yields $[\![\Gamma]\!]_\rho \vDash v : [\![\sigma]\!]_\rho$. Applying it to the second premise and specializing at t yields $[\![\Gamma]\!]_\rho[x \mapsto [\![\sigma]\!]_\rho] \vDash e : t$. By B-LET, $[\![\Gamma]\!]_\rho \vDash \mathrm{let}\ x = v\ \mathrm{in}\ e : t$ holds.

Case SUB. The second premise entails $\rho(\tau) \leq \rho(\tau')$, which implies $[\![\tau]\!]_\rho \supseteq [\![\tau']\!]_\rho$. As a result, $t \in [\![\tau]\!]_\rho$ holds. Thus, applying the induction hypothesis to the first premise yields $[\![\Gamma]\!]_\rho \vDash e : t$.

Case $\forall$ INTRO. Then, $t \in [\![\forall\bar{\alpha}[D].\tau]\!]_\rho$ holds, so there exists an assignment ρ' such that $(\rho \setminus \bar{\alpha} = \rho' \setminus \bar{\alpha}) \wedge \rho' \vdash D$ and $\rho'(\tau) \leq t$. Because $\bar{\alpha} \cap \mathrm{fv}(C) = \varnothing$, $\rho \vdash C$ implies $\rho' \vdash C$. (So, $\rho' \vdash C \wedge D$ holds.) Similarly, because $\bar{\alpha} \cap \mathrm{fv}(\Gamma) = \varnothing$, $[\![\Gamma]\!]_\rho$ equals $[\![\Gamma]\!]_{\rho'}$. Thus, applying the induction hypothesis to the first premise and specializing at t yields $[\![\Gamma]\!]_\rho \vDash e : t$.

Case $\forall$ ELIM. It is clear that $[\![\forall\bar{\alpha}[D].\tau]\!]_\rho \supseteq [\![\tau]\!]_\rho \ni t$. Thus, applying the induction hypothesis to the premise and specializing at t yields $[\![\Gamma]\!]_\rho \vDash e : t$.

Case $\exists$ INTRO. Because $\rho \vdash \exists\bar{\alpha}.C$, there exists an assignment ρ' such that $(\rho \setminus \bar{\alpha} = \rho' \setminus \bar{\alpha}) \wedge \rho' \vdash C$. Because $\bar{\alpha} \cap \mathrm{fv}(\Gamma, \sigma) = \varnothing$, $[\![\Gamma]\!]_{\rho'}$ (resp. $[\![\sigma]\!]_{\rho'}$) coincides with $[\![\Gamma]\!]_\rho$ (resp. $[\![\sigma]\!]_\rho$). Thus, applying the induction hypothesis to the first premise yields $[\![\Gamma]\!]_\rho \vDash e : [\![\sigma]\!]_\rho$.

Case WEAKEN. According to the second premise, $\rho \vdash C$ implies $\rho \vdash C'$. Applying the induction hypothesis to the first premise yields $[\![\Gamma]\!]_\rho \vDash e : [\![\sigma]\!]_\rho$. □

Type soundness for HM(X) is a corollary of theorem 3. It is easy to check that a closed expression e is well-typed under some constraint C and environment Γ if and only if it is well-typed under the **true** constraint and the empty environment $\varnothing$. In such a case, we simply say that e is well-typed. We say that e *goes wrong* if and only if $e/\varnothing \to^\star e'/\sigma'$, where e' is irreducible but not a value, holds.

Theorem 4 (Type Soundness) *If e is a well-typed, closed expression, then e does not go wrong.*

Proof. If e is well-typed, then $\mathbf{true}, \varnothing \vdash e : \sigma$ holds for some type scheme σ. By theorem 3, $\varnothing \vDash e : [\![\sigma]\!]$ holds in B(T). Because $[\![\sigma]\!]$ is a polytype, it is non-empty, so $\varnothing \vDash e : t$ holds

for some $t \in T$. As a result, the initial configuration $e/\varnothing$ is well-typed, i.e. $\vDash e/\varnothing : t$ holds. The result follows by theorems 1 and 2. □

References

[1] Martín Abadi, Benjamin Pierce, and Gordon Plotkin. Faithful ideal models for recursive polymorphic types. *International Journal of Foundations of Computer Science*, 2(1):1–21, March 1991. URL: `http://www.cis.upenn.edu/~bcpierce/papers/ideals.ps`.

[2] Sylvain Conchon and François Pottier. JOIN(X): Constraint-based type inference for the join-calculus. To appear at *ESOP'01*. URL: `http://pauillac.inria.fr/~fpottier/publis/conchon-fpottier-esop01.ps.gz`, April 2001.

[3] David B. MacQueen, Gordon D. Plotkin, and Ravi Sethi. An ideal model for recursive polymorphic types. *Information and Control*, 71(1–2):95–130, October–November 1986.

[4] Robin Milner. A theory of type polymorphism in programming. *Journal of Computer and System Sciences*, 17(3):348–375, December 1978.

[5] Martin Odersky, Martin Sulzmann, and Martin Wehr. Type inference with constrained types. *Theory and Practice of Object Systems*, 5(1):35–55, 1999. URL: `http://www.cs.mu.oz.au/~sulzmann/publications/tapos.ps`.

[6] François Pottier and Vincent Simonet. Information flow inference for ML. To appear, 2001.

[7] Martin Sulzmann. *A general framework for Hindley/Milner type systems with constraints*. PhD thesis, Yale University, Department of Computer Science, May 2000. URL: `http://www.cs.mu.oz.au/~sulzmann/publications/diss.ps.gz`.

[8] Martin Sulzmann, Martin Müller, and Christoph Zenger. Hindley/Milner style type systems in constraint form. Research Report ACRC–99–009, University of South Australia, School of Computer and Information Science, July 1999. URL: `http://www.ps.uni-sb.de/~mmueller/papers/hm-constraints.ps.gz`.

[9] Andrew K. Wright. Simple imperative polymorphism. *Lisp and Symbolic Computation*, 8(4):343–356, December 1995.

[10] Andrew K. Wright and Matthias Felleisen. A syntactic approach to type soundness. *Information and Computation*, 115(1):38–94, November 1994. URL: `http://www.cs.rice.edu/CS/PLT/Publications/ic94-wf.ps.gz`.

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot-St-Martin (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399