

Syntactic Type Soundness for $HM(X)$

Christian Skalka
The Johns Hopkins University

François Pottier
INRIA Rocquencourt

The system $HM(X)$

The system $HM(X)$ is a constraint based type *framework*:

- Sulzmann, PhD thesis 2000
- Odersky, Sulzmann and Wehr, TOPAS 1999, vol. 5 no. 1

The framework provides a type system for functional core that may be *instantiated* with specialized constraint systems for particular applications:

- proven sound with respect to *denotational semantics*
- *semi-syntactic* soundness proof by Pottier, Res. Report 4150, INRIA
 - not purely syntactic; no subject reduction

The system $\text{HM}(X)$

Principal contribution:

- *purely syntactic* result fills a gap in the literature
 - obtained via standard techniques

Other (minor) contributions include:

- addition of *state* to the core language
- addition of *recursive binding mechanism* to core language
- more *direct axiomatization* of *constraint systems*

NB: we focus on *logical* type system, not inference

The HM(X) language

The HM(X) framework provides a *core functional calculus*:

- functional abstractions $\text{fix } z.\lambda x.e$, where z binds to $\text{fix } z.\lambda x.e$ in e
- standard reference operations ref , $!$ and $:=$
- let expressions $\text{let } x = v \text{ in } e$; note *values restriction* to ensure safe interaction of state and polymorphism
- *functional* constants $c \in \text{Const}$
 - The set Const is *defined in instantiations*

Semantics of $\text{HM}(X)$

The behavior of $\text{HM}(X)$ is defined via an *operational semantics*, a reduction relation \rightarrow on *configurations* e/ς :

- *stores* ς are partial mappings from *locations* l to values v
- reduction rule for applications: $(\text{fix } z.\lambda x.e)v/\varsigma \rightarrow e[v/x][\text{fix } z.\lambda x.e/z]/\varsigma$
- reduction rule for functional constants: $\mathbf{c} v/\varsigma \rightarrow \delta(\mathbf{c}, v)/\varsigma$
 - The function δ is *defined in instantiations*
- other reduction rules are standard

The $\text{HM}(X)$ type and constraint language

The $\text{HM}(X)$ framework provides a basic language of types:

$$\tau ::= \alpha \mid \tau \rightarrow \tau \mid \tau \text{ ref}$$

constraints:

$$C ::= \mathbf{true} \mid \tau = \tau \mid \tau \leq \tau \mid C \wedge C \mid \exists \alpha. C$$

and constrained polymorphic type schemes:

$$\sigma ::= \forall \bar{\alpha}[C]. \tau$$

Any instance of $\text{HM}(X)$:

- *extends* the language of types and constraints with specialized terms
- *defines initial type bindings* Δ for constants in $Const$

HM(X) type judgment rules (highlights)

$$\text{SUB} \frac{C, \Gamma \vdash e : \tau \quad C \Vdash \tau \leq \tau'}{C, \Gamma \vdash e : \tau'}$$

$$\text{CONST} \\ C, \Gamma \vdash \mathbf{c} : \Delta(\mathbf{c})$$

$$\text{LET} \frac{C, \Gamma \vdash e_1 : \sigma \quad C, (\Gamma; x : \sigma) \vdash e_2 : \tau}{C, \Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau}$$

$$\text{APP} \frac{C, \Gamma \vdash e_1 : \tau' \rightarrow \tau \quad C, \Gamma \vdash e_2 : \tau'}{C, \Gamma \vdash e_1 e_2 : \tau}$$

$$\forall \text{ELIM} \frac{C, \Gamma \vdash e : \forall \bar{\alpha}[D].\tau' \quad C \Vdash [\bar{\tau}/\bar{\alpha}]D}{C, \Gamma \vdash e : [\bar{\tau}/\bar{\alpha}]\tau'}$$

Interpretation of constraints

We interpret constraints via *assignments* ρ , which map type variables to *monotypes* (variable-free types):

- a *model* (T, \leq) is a partially ordered set of monotypes T , satisfying the usual subtyping properties
- a *standard interpretation* consists of an extension of assignments to arbitrary types, and a *constraint satisfaction relation* $\rho \vdash C$:

$$\begin{aligned}\rho \vdash \tau_1 \leq \tau_2 &\Leftrightarrow \rho(\tau_1) \leq \rho(\tau_2) \\ \rho \vdash C_1 \wedge C_2 &\Leftrightarrow (\rho \vdash C_1) \wedge (\rho \vdash C_2) \\ &\vdots\end{aligned}$$

- we write $C \Vdash C'$ iff for all ρ , if $\rho \vdash C$ then $\rho \vdash C'$

Any instantiation of *must* specify a model and standard interpretation.

The meaning of $\text{HM}(X)$ type soundness

Our type soundness result shows that any *instance of* $\text{HM}(X)$ *automatically* enjoys type soundness in the framework.

Saves significant proof effort! Soundness cases for constants in Const must still be proven, the *δ -typability condition*:

- for every constant c and closed value v , if $C, \Gamma \vdash c : \tau_1 \rightarrow \tau_2$ and $C, \Gamma \vdash v : \tau_1$ hold, then $\delta(c, v)$ is defined and $C, \Gamma \vdash \delta(c, v) : \tau_2$ holds

Trickiest bits of syntactic type soundness are taken care of by our result.

Instance of $\text{HM}(X)$

To sum up, an instance of $\text{HM}(X)$ is defined by:

- an extension of the type and constraint language, together with a standard interpretation
- a particular choice of the set of constants Const , together with functions δ and Δ , meeting the δ -typability condition

Any instance of $\text{HM}(X)$ enjoys syntactic type soundness.

Type soundness: definitions

Given the following standard definition:

Definition 1 *If $e/\emptyset \rightarrow^* e'/\varsigma'$, where e'/ς' is irreducible but e' is not a value, then e is said to go wrong.*

Our aim is to prove the following result:

Theorem 1 (Type Safety) *Let e be an expression in an instance of $HM(X)$; then if e is closed and well-typed, then e does not go wrong.*

Accomplished by proving *subject reduction*.

The proof

To obtain subject reduction, we prove a number of preliminary results.

The first involves type *substitutions* φ , asserting that substitution preserves type derivations:

Lemma 1 (Type Instantiation) *If there exists a derivation of $C, \Gamma \vdash e : \sigma$, then there exists a derivation of $\varphi(C), \varphi(\Gamma) \vdash e : \varphi(\sigma)$ with the same structure.*

Proof: By induction on the derivation of $C, \Gamma \vdash e : \sigma$.

The proof: normalization

The next step in our proof is *normalization*, which will allow us to consider *canonical, syntax-directed* derivations in subject reduction.

We begin by showing that non-syntax directed rules can be “collapsed”:

Lemma 2 *Any two consecutive instances of \forall INTRO and \forall ELIM may be suppressed.*

Proof: By type instantiation Lemma.

Lemma 3 *Any two consecutive instances of SUB may be collapsed into one.*

Proof: By transitivity of \leq .

The proof: normalization

Since the previous Lemmas demonstrate that non-syntax-directed rules may be collapsed, we are able to easily prove our normalization result:

Lemma 4 (Normalization) *If $C, \Gamma \vdash e : \tau$ holds, then it must follow by SUB from a judgement \mathcal{J} such that \mathcal{J} is an instance of a syntax-directed rule corresponding to the form of e .*

Proof: By Lemmas 2 and 3.

The proof: value substitution

Following standard methods, the tricky application and let cases of subject reduction are handled by an auxiliary *substitution* Lemma:

Lemma 5 (Substitution) *If $C, \Gamma; x : \sigma' \vdash e : \sigma$ and $C, \Gamma \vdash v : \sigma'$ then $C, \Gamma \vdash e[v/x] : \sigma$.*

Proof: By induction on the derivation of $C, \Gamma; x : \sigma' \vdash e : \sigma$.

The proof: subject reduction

As a prelude to subject reduction, we extend type judgements to configurations:

$$\text{CONFIG} \quad \frac{C, \Gamma \vdash e : \tau \quad \forall l \in \text{dom}(\Gamma) \quad C, \Gamma \vdash \varsigma(l) : \Gamma(l)}{C, \Gamma \vdash e/\varsigma : \tau}$$

Our subject reduction result is then stated as follows:

Theorem 2 (Subject Reduction) *If $C, \Gamma \vdash e_1/\varsigma_1 : \tau$ is derivable and $e_1/\varsigma_1 \rightarrow e_2/\varsigma_2$, then, for some Γ' which extends Γ with bindings for new memory locations, $C, \Gamma' \vdash e_2/\varsigma_2 : \tau$ is derivable.*

The proof: subject reduction

Proof: By normalization Lemma and case analysis on the reduction: $e_1/s_1 \rightarrow e_2/s_2$

- The case $e_1 = cv$ follows by δ -typability
- The cases $e_1 = (\text{fix } z.\lambda x.e)v$ and let $x = v$ in e follow by substitution Lemma.
- Other cases follow by construction.

The proof: progress

To make the final step to type safety, we prove a stronger *progress* result:

Lemma 6 (Progress) *If a closed configuration e/ς is well-typed and irreducible, then e is a value.*

Proof: By contradiction, via an examination of cases in which e/ς is irreducible and is *not* a value:

- We demonstrate that such configurations are not well-typed

HM(X) type soundness achieved

These results enable us to prove type safety in a straightforward manner:

Theorem 1 (Type Safety) Let e be an expression in an instance of HM(X); then if e is closed and well-typed, then e does not go wrong.

Proof: By induction on the length of the reduction sequence, subject reduction, and progress:

- *subject reduction* shows that reduction *preserves* well-typedness
- *progress* shows that no configuration in reduction sequence can be semantically ill-defined

Conclusion

$HM(X)$ is a constraint based type framework, useful for easy prototyping of novel languages and type systems:

- framework may be *instantiated*:
 - with new language constants and reduction rules
 - with specialized type language for new constants
- instantiations automatically enjoy *syntactic type soundness* in the framework:
 - instantiations must satisfy basic properties
 - our result first purely syntactic soundness result for $HM(X)$

<http://www.cs.jhu.edu/~ces/work.html>