Deriving a Hoare-Floyd logic for non-local jumps from a formulae-as-types notion of control

Tristan Crolard, Emmanuel Polonowski

▶ **To cite this version:**

Tristan Crolard, Emmanuel Polonowski. Deriving a Hoare-Floyd logic for non-local jumps from a formulae-as-types notion of control. The 22nd Nordic Workshop on Programming Theory, Nov 2010, Turku, Finland. pp.70–71. hal-00651455

HAL Id: hal-00651455
https://hal.science/hal-00651455v1

Submitted on 13 Dec 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Deriving a Hoare-Floyd logic for non-local jumps from a formulæ-as-types notion of control

Tristan Crolard
Emmanuel Polonowski
LACL – Université Paris-Est, France
{crolard,polonowski}@u-pec.fr

Abstract

We derive a Hoare-Floyd logic for non-local jumps and mutable higher-order procedural variables from a formulæ-as-types notion of control for classical logic. The main contribution of this work is the design of an imperative dependent type system for non-local jumps which corresponds to classical logic but where the famous *consequence rule* is still derivable.

Hoare-Floyd logics for non-local jumps are notoriously difficult to obtain, especially in the presence of local mutable variables [7]. As far as we know, the question of proving the correctness of imperative programs which combine local mutable *higher-order procedural variables* and non-local jumps has not even been addressed. On the other hand, we know since Griffin's pioneering work [3] how to prove the correctness of (higher-order) functional programs with control in direct style, thanks to the formulæ-as-types interpretation of classical logic.

In [1], Chapter 3, we have thus extended the formulæ-as-types notion of control to imperative programs with higher-order procedural mutable variables and non-local jumps. Our technique, which was inspired by Landin's seminal paper [4], consists in defining an imperative dependent type system **ID** by translation into a functional dependent type system (which is actually Leivant's **ML1P** [5]). This imperative language, called LOOP$^{\omega}$, was defined by the authors in [2].

Similarly to **ML1P**, the imperative type system is parametrized by a first-order signature and an equational system $\mathscr{E}$ which defines a set of functions in the style of Herbrand-Gödel. The syntax of imperative types of **ID** (with dependent procedure types and dependent records) is the following:

$$\sigma,\tau ::= \mathbf{nat}(n) \mid \mathbf{proc}\ \forall\vec{\imath}(\mathbf{in}\ \vec{\tau};\mathbf{out}\ \vec{\sigma}) \mid \exists\vec{\imath}(\sigma_1,\ldots,\sigma_n) \mid n=m$$

Typing judgements of **ID** have the form $\Gamma;\Omega \vdash e : \psi$ if e is an expression and $\Gamma;\Omega \vdash s \rhd \Omega'$ if s is a sequence, where environments Γ and Ω corresponds respectively to immutable and mutable variables. Note that our type system is *pseudo-dynamic* in the sense that the type of mutable variables can change in a sequence and the new types are given by Ω' (as in [8]). For instance, here is the typing rule of the **for** loop:

$$\frac{\Gamma;\Omega,\vec{x}:\vec{\sigma}[\mathbf{0}/i] \vdash e : \mathbf{nat}(n) \qquad \Gamma,y:\mathbf{nat}(i);\vec{x}:\vec{\sigma} \vdash s \rhd \vec{x}:\vec{\sigma}[\mathbf{s}(i)/i]}{\Gamma;\Omega,\vec{x}:\vec{\sigma}[\mathbf{0}/i] \vdash \mathbf{for}\ y := 0\ \mathbf{until}\ e\ \{s\}_{\vec{x}} \rhd \vec{x}:\vec{\sigma}[n/i]}$$

Embedding a Hoare-Floyd logic

It is almost straightforward to embed a Hoare-Floyd logic into **ID**. Indeed, let us take a global mutable variable, dubbed ***assert***, and let us assume that this global variable is simulated in the usual *state-passing style* (the variable is passed as an explicit **in** and **out** parameter to each procedure call). Consequently, any sequence shall be typed with a sequent of the form $\Gamma;\Omega,\textbf{\textit{assert}} : \varphi \vdash s \rhd \Omega',\textbf{\textit{assert}} : \psi$. If we now introduce the usual Hoare notation for triples (which hides the name of global variable ***assert***), we obtain judgments of the form $\Gamma;\Omega \vdash \{\varphi\}s \rhd \Omega'\{\psi\}$. Rules very similar to Hoare rules are then derivable: for instance, the type of ***assert*** corresponds to the invariant in a loop, and to the type of *pre*

and *post* conditions in a procedure type. The only rule which is not directly derivable is the well-known *consequence rule*:

$$\frac{\Gamma,\Omega \vdash \varphi' \Rightarrow \varphi \qquad \Gamma;\Omega \vdash \{\varphi\}s \rhd \Omega'\{\psi\} \qquad \Gamma,\Omega \vdash \psi \Rightarrow \psi'}{\Gamma;\Omega \vdash \{\varphi'\}s \rhd \Omega'\{\psi'\}}$$

This rule deserves a specific treatment since no proof-term is required for the proof obligations. However, it is well-known that in intuitionistic logic the proof of some formulas have no computational content (they are called *data-mute* in [5]). The consequence rule is thus derivable if we restrict (without loss of generality) the set of assertions to data-mute formulas.

Non-local jumps

The imperative language was then extended in [1] with labels and non-local jumps. At the (dependent) type level, this extension (called $\mathbf{ID}^c$) corresponds to an extension from intuitionistic logic to classical logic. For instance, the following typing rules for labels and jumps are derivable (where first-class labels are typed by the negation):

$$\frac{\Gamma,k:\neg\vec{\sigma};\vec{z}:\vec{\tau} \vdash s \rhd \vec{z}:\vec{\sigma} \qquad \Gamma;\Omega,\vec{z}:\vec{\sigma} \vdash s' \rhd \Omega'}{\Gamma;\Omega,\vec{z}:\vec{\tau} \vdash k:\{s\}_{\vec{z}};\ s' \rhd \Omega'} \qquad \frac{\Gamma;\Omega,\vec{z}:\vec{\tau} \vdash k:\neg\vec{\sigma} \qquad \Gamma;\Omega,\vec{z}:\vec{\tau} \vdash \vec{e}:\vec{\sigma}}{\Gamma;\Omega,\vec{z}:\vec{\tau} \vdash \mathbf{jump}(k,\vec{e})_{\vec{z}} \rhd \vec{z}:\vec{\tau}'}$$

However, deriving a Hoare-Floyd logic for non-local jumps is not straightforward since there is no obvious notion of *data-mute* formula in classical logic (as noted also in [6]), and thus the consequence rule is in general not derivable. The problem comes from the fact that, in presence of control operators, the proof-terms corresponding to proof-obligations may interact with the program. We shall exhibit an example of such program and we shall present a general solution to this problem which relies on the distinction between purely functional terms and imperative procedures (possibly containing non-local jumps).

References

[1] T. Crolard. Certification de programmes impératifs d'ordre supérieur avec mécanismes de contrôle. Habilitation Thesis. LACL, Université Paris-Est, 2010.

[2] T. Crolard, E. Polonowski, and P. Valarcher. Extending the Loop Language with Higher-Order Procedural Variables. *Special issue of ACM TOCL on Implicit Computational Complexity*, 10(4):1–37, 2009.

[3] T. G. Griffin. A formulæ-as-types notion of control. In *Conference Record of the 17th Annual ACM Symposium on Principles of Programming Langages*, pages 47–58, 1990.

[4] P. J. Landin. A correspondence between ALGOL 60 and Church's lambda-notation: part I. *Commun. ACM*, 8(2):89–101, 1965.

[5] D. Leivant. Contracting proofs to programs. In Odifreddi, editor, *Logic and Computer Science*, pages 279–327. Academic Press, 1990.

[6] Y. Makarov. Practical Program Extraction from Classical Proofs. *Electronic Notes in Theoretical Computer Science*, 155:521 – 542, 2006. Proceedings of the 21st Annual Conference on Mathematical Foundations of Programming Semantics (MFPS XXI).

[7] R. D. Tennent and J. K. Tobin. Continuations in Possible-World Semantics. *Theor. Comput. Sci.*, 85(2):283–303, 1991.

[8] H. Xi. Imperative Programming with Dependent Types. In *Proceedings of 15th IEEE Symposium on Logic in Computer Science*, pages 375–387, Santa Barbara, June 2000.