Type systems for programming languages

Didier Rémy

Academic year 2020-2021
Version of October 20, 2020
# Contents

1 Introduction ............................................................. 7
  1.1 Overview of the course ......................................... 7
  1.2 Requirements ..................................................... 9
  1.3 About Functional Programming ................................. 9
  1.4 About Types ...................................................... 9
  1.5 Acknowledgment .................................................. 11

2 The untyped $\lambda$-calculus .................................... 13
  2.1 Syntax ................................................................... 13
  2.2 Semantics .......................................................... 15
    2.2.1 Strong v.s. weak reduction strategies .................. 15
    2.2.2 Call-by-value semantics .................................... 16
  2.3 Answers to exercises .............................................. 18

3 Simply-typed lambda-calculus ..................................... 21
  3.1 Syntax ................................................................... 21
  3.2 Dynamic semantics ............................................... 21
  3.3 Type system ........................................................ 22
  3.4 Type soundness .................................................... 25
    3.4.1 Proof of subject reduction ................................. 26
    3.4.2 Proof of progress ............................................. 28
  3.5 Simple extensions ............................................... 30
    3.5.1 Unit .............................................................. 30
    3.5.2 Boolean ........................................................ 30
    3.5.3 Pairs ........................................................... 31
    3.5.4 Sums ............................................................. 32
    3.5.5 Modularity of extensions .................................... 32
    3.5.6 Recursive functions .......................................... 33
    3.5.7 A derived construct: let-bindings ....................... 33
  3.6 Exceptions ........................................................ 35
    3.6.1 Semantics ...................................................... 35
Chapter 8
Logical Relations

8.1 Introduction

Logical relations are relations between well-typed programs defined inductively on the structure of types.

There are two kinds of logical relations: unary and binary. Unary relations are predicate on expressions, while binary relations relates two expressions of the same type.

What they can be used for: unary relations can be used to prove type safety and strong normalisation; binary relations can be used to prove equivalence of programs and non-interference properties.

Logical relations are a common proof method for programming language researchers that every one ought to know.

8.2 Normalization of simply-typed $\lambda$-calculus

In general, types also ensure termination of programs—as long as no form of recursion in types or terms has been added. Even if one wishes to add recursion explicitly later on, it is an important property of the design that non-termination is originating from the constructs for recursion only and could not occur without it.

The simply-typed $\lambda$-calculus is also lifted at the level of types in richer type systems such as System $F^\omega$; then, the decidability of type-equality depends on the termination of the reduction at the type level.

Proving termination of reduction in fragments of the $\lambda$-calculus is often a difficult task because reduction may create new redexes or duplicate existing ones. However, the proof of termination for the simply-typed $\lambda$-calculus is simple enough and interesting to be presented here. Notice that our presentation of simply-typed $\lambda$-calculus is equipped with a call-by-value semantics, while proofs of termination are usually done with a strong evaluation strategy where reduction can occur in any context.
We follow the proof schema of Pierce (2002), which is a modern presentation in a call-by-value setting of an older proof by Hindley and Seldin (1986). The proof method, which is now a standard one, is due to Tait (1967). It consists in first building the set $T_\tau$ of terminating closed terms of type $\tau$, and then showing that any term of type $\tau$ is actually in $T_\tau$, by induction on terms. Unfortunately, stated as such, this hypothesis is too weak. The difficulty in such cases is usually to find a strong enough induction hypothesis. The solution in this case is to require that terms in $T_{\tau_1 \to \tau_2}$ not only terminate but also terminate when applied to any term in $T_{\tau_1}$.

**Definition 4** Let $T_\tau$ be defined inductively on $\tau$ as follows: let $T_\alpha$ be the set of closed terms that terminates; let $T_{\tau_2 \to \tau_1}$ be the set of (closed) terms $M_1$ of type $\tau_2 \to \tau_1$ that terminates and such that $M_1 M_2$ is in $T_{\tau_1}$ for any (closed) term $M_2$ in $T_{\tau_2}$.

The set $T_\tau$ can be seen as a predicate, i.e. a unary relation. It is called a (unary) logical relation because it is defined inductively on the structure of types. The following proof is then schematic of the use of logical relations.

We state two obvious lemmas to prepare for the main proof. All terms in $T_\tau$ terminate, by definition of $T_\tau$:

**Lemma 37** For any type $\tau$, the reduction of any term in $T_\tau$ halts.

Reduction of closed terms of type $\tau$ preserves membership in $T_\tau$:

**Lemma 38** If $\emptyset \vdash M : \tau$ and $M \to M'$, then $M \in T_\tau$ iff $M' \in T_\tau$.

(Proof p. 171)

Therefore, it just remains to show that any term of type $\tau$ is in $T_\tau$:

**Lemma 39** If $\emptyset \vdash M : \tau$, then $M \in T_\tau$.

The proof is by induction on (the typing derivation of) $M$. However, the case for abstraction requires some similar statement, but for open terms. We need to strengthen the lemma. Actually, to avoid considering open terms, we instead require the statement to hold for all closed instances of an open term:

**Lemma 40 (strengthened)** If $(x_i : \tau_i)^{ieI} \vdash M : \tau$, then for any closed values $(V_i)^{ieI}$ in $(T_{\tau_i})^{ieI}$, the term $[(x_i \mapsto V_i)^{ieI}]M$ is in $T_\tau$.

**Proof:** We write $\Gamma$ for $(x_i : \tau_i)^{ieI}$ and $\theta$ for $[(x_i \mapsto V_i)^{ieI}]$. Assume $\Gamma \vdash M : \tau$ (1) and $(V_i)^{ieI}$ in $(T_{\tau_i})^{ieI}$ (2). We show that $\theta M$ is in $T_\tau$ (3) by structural induction on $M$.

*Case $M$ is $x_i$: Immediate since the conclusion (3) is one of the hypotheses (2).*

*Case $M$ is $M_1 M_2$: By inversion of the typing judgment (1), we have $\Gamma \vdash M_1 : \tau_2 \to \tau$ (4) and $\Gamma \vdash M_2 : \tau_2$ (5) for some type $\tau_2$. Therefore, by induction hypothesis applied to (4) and (5),
we have $\theta M_1 \in T_{\tau_2} \rightarrow \tau$ and $\theta M_2 \in T_{\tau_2}$. Thus, by definition of $T_{\tau}$, we have $(\theta M_1) (\theta M_2) \in T_{\tau}$; that is, $\theta M \in T_{\tau}$.

Case $M$ is $\lambda x: \tau_1. M_2$: By inversion of the typing judgment (1), we have $\Gamma, x: \tau_1 \vdash M_2 : \tau_2$ (6) where $\tau_1 \rightarrow \tau_2$ is $\tau$ (7). Since $M$ is a value, it is terminating. Hence, to ensure (3), it suffices to show that the application of $\theta M$ to any $M_1$ in $T_{\tau_1}$ is in $T_{\tau_2}$ (8). Let $M_1 \in T_{\tau_1}$. By definition of $T_{\tau_1}$, the term $M_1$ reduces to some value $V$, which by subject reduction has type $\tau_1$, and so is in $T_{\tau_1}$ (9). We have:

$$(\theta M) M_1 \triangleq (\theta (\lambda x: \tau_1. M_2)) M_1 \quad \text{by definition of } M$$

$$= (\lambda x: \tau_1. \theta M_2) M_1 \quad \text{choose } x \neq \bar{x}$$

$$\rightarrow^* (\lambda x: \tau_1. \theta M_2) V \quad \text{by (9)}$$

$$\rightarrow [x \mapsto V] (\theta M_2) \quad \text{by (\beta)}$$

$$= ([x \mapsto V] \theta) M_2 \quad \epsilon T_{\tau_2} \quad \text{by I.H.}$$

In the last step, we may apply the induction hypothesis, since the first hypothesis is (6) and the second one follows from (2) and (9). In summary, $(\theta M) M_1$ reduces to a term in $T_{\tau_2}$. Since $T_{\tau_2}$ is closed by reduction, $(\theta M) M_1$ itself in in $T_{\tau_2}$, which establishes (8), as expected.

8.3 Proofs and Solution to Exercises

Proof of Lemma 38

By induction on the structure of the type $\tau$.

Case $\tau$ is $\alpha$: Then $T_{\tau}$ is the set of terms that terminates. If $M \rightarrow M'$, the termination of $M$, i.e. $M \in T_{\alpha}$, is equivalent to the termination of $M'$, i.e. $M' \in T_{\alpha}$.

Case $\tau$ is $\tau_1 \rightarrow \tau_2$: Then $T_{\tau}$ is the set of type $\tau$ that terminate and also terminate when applied to any term $M_1$ of type $\tau_1$. Assume $\varnothing \vdash M : \tau$ (1) and $M \rightarrow M'$ (2). By subject reduction, we have $\varnothing \vdash M' : \tau$. Moreover, from (2), termination of $M$ and termination of $M'$ are equivalent. Therefore, it only remains to check that for any term $M_1$ of $T_{\tau_1}$, $M M_1$ and $M' M_1$ are both in $T_{\tau_2}$ or both outside of $T_{\tau_2}$ (3). Let $M_1$ be in $T_{\tau_1}$. We have $\varnothing \vdash M_1 : \tau$ and thus $\varnothing \vdash M M_1 : \tau_2$. We also have the call-by-value reduction $M M_1 \rightarrow M' M_1$, Hence, (3) follows by induction hypothesis.
Bibliography

▷ A tour of scala: Implicit parameters. Part of scala documentation.


