IxFree

Step-Indexed Logical Relations in Coq

Piotr Polesiuk

Institute of Computer Science, University of Wrocław

ppolesiuk@cs.uni.wroc.pl

Abstract

Step-indexed logical relations allow to handle complex programming language features, but using them in practice leads to complicated proofs because of ubiquitous step-index arithmetic. Dreyer et al. proposed the LSLR logic to treat indices in a more abstract way hiding them from the user. In this work we present IxFree, a Coq library, based on a shallow embedding of the LSLR logic, to reason about step-indexed logical relations in a simple and elegant way.

1. Introduction

Step-indexed logical relations [1–3] are a powerful tool for reasoning about programming languages. Instead of describing a general behavior of program execution, they focus on the first n computation steps, where the step index n is an additional parameter of the relation. This additional parameter makes it possible to define logical relations inductively not only on the structure of types, but also on the number of computation steps that are allowed for a program to make and, therefore, they provide an elegant way to reason about features that introduce non-termination to the programming language, including recursive types [2] and references [1].

However, reasoning directly about step-indexed logical relations is tedious because proofs become obscured by step-index arithmetic. In order to avoid this problem, Dreyer et al. proposed logical step-indexed logical relations (LSLR) [6] that is a modal logic where step-indices are treated in a more abstract way and they are hidden from the user. The key feature of the LSLR logic is a *later modality* borrowed from Appel et al.'s "very modal model" [4] that allows to shift relation to another index and helps defining recursive predicates over indices. Since LSLR is a modal logic, it cannot be used directly in Coq without proper embedding.

In this work we present IxFree: a shallow embedding of LSLR logic in Coq. In contrast to Appel et al.'s formalization of the "very modal model," our approach is less shallow, which allows us to reuse a number of existing Coq tactics. We used our library to formalize step-indexed logical relations for coherence of control-effect subtyping [5]. The source code is available at `http://www.ii.uni.wroc.pl/~ppolesiuk/IxFree`.

2. Embedding of the LSLR logic

The LSLR logic is an intuitionistic logic for reasoning about one particular Kripke model: where possible worlds are natural numbers (step-indices) and where future worlds have smaller indices than the present one. All formulas are interpreted as monotone (non-increasing) sequences of truth values, whereas the connectives are interpreted as usual. In particular, in the case of implication we quantify over all future worlds to ensure monotonicity, so the formula $\varphi \Rightarrow \psi$ is valid at index n (written $n \models \varphi \Rightarrow \psi$) iff $k \models \varphi$ implies $k \models \psi$ for every $k \leq n$. In contrast to the original formulation we do not assume that all formulas are valid in world 0, because it is not necessary.

2.1 Indexed propositions

To represent propositions in the IxFree library we have a special type `IProp` of "indexed propositions" defined as a type of monotone functions from `nat` to `Prop`.

```
Definition monotone (P : nat → Prop) :=
  ∀ n, P (S n) → P n.
Definition IProp :=
  { P : nat → Prop | monotone P }.

Definition I_valid_at (n : nat) (P : IProp) :=
  proj1_sig P n.
Notation "n ⊨ P" := (I_valid_at n P).
```

Logical connectives are functions on type `IProp` with defined human readable notation. The library provides lemmas and tactics representing the most important inference rules. Tactics not only apply the corresponding lemmas, but also hide the step index arithmetic from the user. For instance, when proving sequent $Q \vdash P \Rightarrow Q$ represented by the following Coq goal

```
P : IProp
Q : IProp
k : nat
H1 : k ⊨ Q
============================
k ⊨ P ⇒ Q
```

the introduction of implication tactic `iintro H2` behaves exactly like introduction of implication rule, producing the goal

```
P : IProp
Q : IProp
k : nat
H1 : k ⊨ Q
H2 : k ⊨ P
============================
k ⊨ Q
```

even if the lemma corresponding to that rule requires quantification over all smaller indices:

```
Lemma I_arrow_intro {n : nat} {P Q : IProp} :
  (∀ k, k ≤ n, (k ⊨ P) → (k ⊨ Q)) →
  (n ⊨ P ⇒ Q).
```

2.2 Later modality

The LSLR logic is equipped with a modal operator $\triangleright$ (later), to provide access to strictly future worlds. The formula $\triangleright\varphi$ means φ *holds in any future world*, or formally $\triangleright\varphi$ is always valid at world 0, and $n + 1 \models \triangleright\varphi$ iff φ is valid at n (and other future worlds by monotonicity). The later operator comes with two inference rules:

$$\frac{\Gamma, \Sigma \vdash \varphi}{\Gamma, \triangleright\Sigma \vdash \triangleright\varphi}\ \triangleright\text{-intro} \qquad \frac{\Gamma, \triangleright\varphi \vdash \varphi}{\Gamma \vdash \varphi}\ \text{Löb}$$

The first rule allows one to shift reasoning to a future world, making the assumptions about the future world available. The Löb rule expresses an induction principle for indices. Note that the premise of the rule also captures the base case, because the assumption $\triangleright\varphi$ is trivial in the world 0. The later operator comes with no general elimination rule.

As other connectives in the IxFree library, later is represented as a function on type `IProp` and the library provides tactics that correspond to the inference rules. In particular, tactic `later_shift` corresponding to the $\triangleright$-intro rule not only removes the later operator from the goal, but also from the premises that it guards.

2.3 Recursive predicates

Predicates in LSLR logic as well as step-indexed logical relations can be defined inductively on indices. More generally, we can define a recursive predicate $\mu r.\varphi(r)$, provided all occurrences of r in φ are guarded by the later operator, to guarantee well-foundedness of the definition. Such syntactic requirement is not compatible with structural recursion in Coq, so we rely on the notion of *contractiveness* [4]. Informally, function is contractive if it maps approximately equal arguments to more equal results. This intuition can be expressed using the later modality:

```
Definition contractive (l : list Type)
    (f : IRel l → IRel l) : Prop :=
  ∀ R₁ R₂, ⊨ ▷(R₁ ≈ᵢ R₂) ⇒ f R₁ ≈ᵢ f R₂.
```

where `IRel l` is a type of indexed relations on types described by `l`, and $\approx_i$ is an indexed version of relation equivalence. The library provides a general method of constructing recursive relations as a fixed point of a contractive function:

```
Definition I_fix
    (l : list Type) (f : IRel l → IRel l) :
  contractive l f → IRel l.
```

If all occurrences of the function argument are guarded by the later operator, then the function can be proven to be contractive, and the proof can be (mostly) automatized by the `auto_contr` tactic.

2.4 User defined predicates

Since the logic is developed for reasoning about one particular model, we can freely add new inference rules for the logic if we prove they are valid in the model. We can also add new relations or predicates to the logic if we provide their monotone interpretation.

```
Definition mk_IProp (P : nat → Prop) :
  monotone P → IProp.
```

In particular, constant functions are monotone, so we can safely use predicates defined outside of the logic, such as typing or reduction relations.

```
Definition I_Prop (P : Prop) : IProp :=
  mk_IProp (λ _, P) (λ _ H, H).
Notation "( P )ᵢ" := (I_Prop P).
```

3. Related work

"Very modal model" The Coq formalization accompanying Appel et al.'s "very modal model" [4] hides step-indices in a similar way. But one of the main differences between our library and their formalization is a way of keeping track of the assumptions. Instead of interpreting a sequent $\varphi_1, \ldots, \varphi_n \vdash \psi$ directly, we treat it as $k \models \psi$ with the standard Coq assumptions $k \models \varphi_1, \ldots, k \models \varphi_n$. This approach is very convenient since it allows for reusing a number of existing Coq tactics.

ModuRes ModuRes [7] is a library that also supports step-indexed logical relations, by providing machinery for solving recursive domain equations like recursive predicates in LSLR. But this mechanism is not limited to types representing propositions and because of this generality, step-index arithmetic is not perfectly hidden from the user. We believe that our approach is more convenient in cases where only indexed propositions are needed.

References

[1] A. Ahmed, D. Dreyer, and A. Rossberg. State-dependent representation independence. In B. C. Pierce, editor, *Proceedings of the Thirty-Sixth Annual ACM Symposium on Principles of Programming Languages*, pages 340–353, Savannah, GA, USA, Jan. 2009. ACM Press.

[2] A. J. Ahmed. Step-indexed syntactic logical relations for recursive and quantified types. In P. Sestoft, editor, *Programming Languages and Systems, 15th European Symposium on Programming, ESOP 2006*, volume 3924 of *Lecture Notes in Computer Science*, pages 69–83, Vienna, Austria, Mar. 2006. Springer.

[3] A. W. Appel and D. McAllester. An indexed model of recursive types for foundational proof-carrying code. *ACM Transactions on Programming Languages and Systems*, 23(5):657–683, 2001.

[4] A. W. Appel, P. Melliès, C. D. Richards, and J. Vouillon. A very modal model of a modern, major, general type system. In M. Felleisen, editor, *Proceedings of the Thirty-Fourth Annual ACM Symposium on Principles of Programming Languages*, pages 109–122, Nice, France, Jan. 2007. ACM Press.

[5] D. Biernacki and P. Polesiuk. Logical relations for coherence of effect subtyping. In T. Altenkirch, editor, *13th International Conference on Typed Lambda Calculi and Applications (TLCA 2015)*, volume 38 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 107–122, Warsaw, Poland, July 2015. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik.

[6] D. Dreyer, A. Ahmed, and L. Birkedal. Logical step-indexed logical relations. *Logical Methods in Computer Science*, 7(2:16):1–37, 2011.

[7] F. Sieczkowski, A. Bizjak, and L. Birkedal. Modures: A coq library for modular reasoning about concurrent higher-order imperative programming languages. In C. Urban and X. Zhang, editors, *Interactive Theorem Proving: 6th International Conference, ITP 2015, Nanjing, China, August 24-27, 2015, Proceedings*, pages 375–390, Cham, 2015. Springer International Publishing. ISBN 978-3-319-22102-1.