Proof-relevant rewriting strategies in Coq

Matthieu Sozeau

Inria Paris & PPS, Univ Paris Diderot (UMR-CNRS 7126)
matthieu.sozeau@inria.fr

Abstract

We introduce various enhancements of the generalized rewriting system of Coq. First, we show how the framework can be lifted to proof-relevant relations using the newly introduced universe polymorphic definitions in Coq. Second, we introduce rewriting strategies as a monadic combinator library on top of this framework, resembling the LogicT monad for proof-search (i.e., with backtracking, well-behaved choice and composition). These new features combine to provide a general tool for fine-tuned automated rewriting applicable not only to propositional relations but also general computational type-valued relations. Last, we will also present an idea to handle dependent rewriting, e.g. the ability to rewrite in the domain of a dependent product and get corresponding transportation obligations.

1 Proof-relevant rewriting

The new universe polymorphic extension of Coq [1] is crucial to allow a straightforward generalization of the generalized rewriting framework in Coq [2]. Indeed, the basic notion of proper morphism was previously defined only on propositional relations:

```
Class Proper {A : Type} (R : A → A → Prop) (x : A) :=
  proper_prf : R x x.
```

Although this definition could be written with a Type-valued relation instead in previous versions, generic lemmas about Proper would immediately fix the universe level and would force all relations for which rewriting was used to live at the same level, i.e., a no-go. With universe polymorphism, Proper can be made polymorphic on the $\mathtt{Type}_l$ codomain of the relation and similarly generic lemmas can be instantiated at arbitrary, potentially unrelated levels (for example, Proper itself can be shown to be proper for equivalent relations). Examples of useful proof-relevant rewriting relations abound: the appartness predicate of the reals in the CoRN library, the paths relation of Homotopy Type Theory, and in general, Hom-types of categories are naturally formalized using Type-valued relations.

2 Rewriting strategies

The generalized rewriting tactic is based on a monolithic function that folds through a term and produces type-class constraints for showing that constants are morphisms and applies the rewriting at the appropriate places.

We propose a generalization of this design based on a set of strategies that can be combined to obtain custom rewriting procedures. Its set of strategies is based on Elan's rewriting strategies [3]. Rewriting strategies are applied using the tactic `rewrite_strat` s where s is a strategy expression (already part of Coq 8.4, although undocumented). Strategies are defined inductively as described by the grammar given in Figure 1. Actually a few of these are defined in term of the others using a primitive fixpoint operator:

$$
\begin{array}{lcl}
\mathtt{try}\ s & = & s \mid\mid \mathtt{id} \\
\mathtt{any}\ s & = & \mathtt{fix}\ u.\mathtt{try}\ (s\ ;\ u) \\
\mathtt{repeat}\ s & = & s\ ;\ \mathtt{any}\ s \\
\mathtt{bottomup}\ s & = & \mathtt{fix}\ bu.((\mathtt{progress}\ (\mathtt{subterms}\ bu)) \mid\mid s)\ ;\ \mathtt{try}\ bu \\
\mathtt{topdown}\ s & = & \mathtt{fix}\ td.(s \mid\mid (\mathtt{progress}\ (\mathtt{subterms}\ td)))\ ;\ \mathtt{try}\ td \\
\mathtt{innermost}\ s & = & \mathtt{fix}\ i.((\mathtt{subterm}\ i) \mid\mid s) \\
\mathtt{outermost}\ s & = & \mathtt{fix}\ o.(s \mid\mid (\mathtt{subterm}\ o))
\end{array}
$$

s, t, u	::=	(`<-`)? c	(right to left?) lemma
	|	`fail` | `id`	failure | identity
	|	`refl`	reflexivity
	|	`progress` s	progress
	|	`try` s	failure catch
	|	s `;` u	composition
	|	s `||` t	left-biased choice
	|	`repeat` s	iteration (+)
	|	`any` s	iteration (*)
	|	`subterm(s)?` s	one or all subterms
	|	`innermost` s	innermost first
	|	`outermost` s	outermost first
	|	`bottomup` s	bottom-up
	|	`topdown` s	top-down
	|	`hints` *hintdb*	apply first matching hint
	|	`terms` $c \ldots c$	any of the terms
	|	`eval` *redexpr*	apply reduction
	|	`fold` c	fold expression
	|	`pattern` p	pattern matching

Figure 1: Rewriting strategy syntax

Apart from the basic control strategies, we have lemmas strategies allowing to apply any of a set of rewrite rules (`hints`, `terms`), an evaluation strategy that applies anywhere and reduces according to a selection of the usual $\beta\delta\iota\zeta$-laws of CIC, and a `fold` strategy that can refold constants up to unification. The special `pattern` strategy succeeds only when the term pattern-matches its argument, allowing selective rewriting. With these combinators, we can subsume a few existing tactics of Coq: variants of `rewrite`, `autorewrite`, `eval`/`unfold` and `fold`, in the generalized rewriting setting. We also allow user-defined, fine-tuned strategies, whose performance improves on the `repeat rewrite` strategy implemented by the `autorewrite` tactic for example.

On the implementation side, these strategies are implemented using a success-failure continuation monad, similar to the `LogicT` monad [4], which has efficient backtracking and clear semantics while keeping the code modular. In particular, the combination with universe polymorphism was easy to add thanks to this modularity and the hiding of state passing which is required when manipulating polymorphic constants in the ML tactic.

3 Conclusion

These new features make generalized rewriting a viable framework to work with proof-relevant relations, in particular the paths type of Homotopy Type Theory, and other proof-relevant notions of equality like isomorphisms of types or appartness in real number formalizations. We hope to apply the enhanced tactic to real-world formalizations in the future.

References

[1] Sozeau, M., Tabareau, N.: Universe Polymorphism in Coq. In: ITP'14. (2014) To appear.

[2] Sozeau, M.: A New Look at Generalized Rewriting in Type Theory. Journal of Formalized Reasoning **2**(1) (December 2009) 41–62

[3] Luttik, S.P., Visser, E.: Specification of Rewriting Strategies. In: 2nd International Workshop on the Theory and Practice of Algebraic Specifications (ASF+SDF'97), Electronic Workshops in Computing, Springer-Verlag (1997)

[4] Kiselyov, O., Shan, C.c., Friedman, D.P., Sabry, A.: Backtracking, interleaving, and terminating monad transformers: (functional pearl). In: ICFP '05: Proceedings of the tenth ACM SIGPLAN international conference on Functional programming, New York, NY, USA, ACM (2005) 192–203