LiFtEr: Language to Encode Induction Heuristics

Yutaka Nagashima[1][2]

[1] CIIRC, Czech Technical University in Prague,
Prague, Czech Republic
[2] Department of Computer Science, University of Innsbruck,
Innsbruck, Tyrol, Austria

Abstract

Proof assistants, such as Isabelle/HOL, offer tools to facilitate inductive theorem proving. Isabelle experts know how to use these tools effectively; however, there is a little tool support for transferring this expert knowledge to a wider user audience. To address this problem, we present our domain-specific language, `LiFtEr`. `LiFtEr` allows experienced Isabelle users to encode their induction heuristics in a style independent of any problem domain. `LiFtEr`'s interpreter mechanically checks if a given application of induction tool matches the heuristics, thus automating the knowledge transfer loop.

1 Induction in Isabelle/HOL

Isabelle offers the `induct` proof method to handle inductive problems. Proof methods are the Isar syntactic layer of LCF-style tactics. For example, consider the following reverse functions, `rev` and `itrev`, from literature [3]:

```
primrec rev::"'a list =>'a list" where
  "rev  []      = []"
| "rev (x # xs) = rev xs @ [x]"
fun itrev::"'a list =>'a list =>'a list" where
  "itrev  []    ys = ys"
| "itrev (x#xs) ys = itrev xs (x#ys)"
```

where `#` is the list constructor, and `@` appends two lists into one. One can prove the equivalence of these reverse functions in multiple ways using the `induct` method:

```
lemma prf:"itrev xs ys = rev xs @ ys" apply(induct xs ys rule:itrev.induct) by auto
```

`prf` applies functional induction on `itrev` by passing an auxiliary lemma, `itrev.induct`, to the `rule` field. There are other lesser-known techniques to handle difficult inductive problems using the `induct` method, and sometimes users have to develop useful auxiliary lemmas manually; however, for most cases the problem of how to apply induction boils down to the the following question: *what arguments do you pass to the `induct` method?*

Isabelle experts often apply induction heuristics to answer this question and decide what arguments to pass to the `induct` method; however, they did not have a systematic way to encode such heuristics, which made it difficult for new users to learn how to apply induction effectively.

2 LiFtEr: Language to Encode Induction Heuristics

We address this problem with our domain-specific language, `LiFtEr`. `LiFtEr` allows experienced Isabelle users to encode their induction heuristics in a style independent of problem domains.

LiFtEr's interpreter mechanically checks if a given application of induction is compatible with the induction heuristics written by experienced users.

We designed LiFtEr to encode induction heuristics as assertions on invocations of the induct method in Isabelle. An assertion written in LiFtEr takes a triple of a proof goal at hand, its underlying proof state, and the arguments passed to the induct method to prove the goal. When one applies a LiFtEr assertion to an invocation of the induct method, LiFtEr's interpreter returns a boolean value as the result of the assertion applied to the triple.

The goal of a LiFtEr programmer is to write assertions that implement reliable heuristics. A heuristic encoded as a LiFtEr assertion is reliable when it satisfies the following two properties: first, the LiFtEr interpreter is likely to evaluate the assertion to true when the arguments of the induct method are appropriate for the given proof goal. Second, the interpreter is likely to evaluate the assertion to false when the arguments are inappropriate for the goal.

The following is an example assertion written in LiFtEr:

```
  ∃ r1 : rule. True
→
  ∃ r1 : rule.
    ∃ t1 : term.
      ∃ to1 : term_occurrence ∈ t1 : term.
          r1 is_rule_of to1
        ∧
          ∀ t2 : term ∈ induction_term.
            ∃ to2 : term_occurrence ∈ t2 : term.
              ∃ n : number.
                  is_nth_argument_of (to2, n, to1)
                ∧
                  t2 is_nth_induction_term n
```

As a whole this LiFtEr assertion checks if the following holds: if there exists a rule, $r1$, in the rule field of the induct method, then there exists a term $t1$ with an occurrence $to1$, such that $r1$ is derived by Isabelle when defining $t1$, and for all induction terms $t2$, there exists an occurrence $to2$ of $t2$ such that, there exists a number n, such that $to2$ is the nth argument of $to1$ and that $t2$ is the nth induction terms passed to the induct method.

prf is compatible with this heuristic: there is an argument, itrev.induct, in the rule field, and the occurrence of its related term, itrev, in the proof goal takes all the induction terms, xs and ys, as its arguments in the same order.

3 Conclusion

We presented LiFtEr and its example assertion. LiFtEr is a domain-specific language in the sense that we developed LiFtEr to encode induction heuristics; however, heuristics written in LiFtEr are usually not specific to any problem domain, because LiFtEr's language construct is not specific to any variable names, types, or constants. This absence encourages LiFtEr users to encode heuristics that are not specific to any problem domains but are applicable to many domains. To the best of our knowledge, LiFtEr is the first domain-specific language that allows us to encode induction heuristics as programs. We released a working prototype of the LiFtEr interpreter and six example assertions at GitHub [2]. And a more comprehensive explanation of LiFtEr's grammar is provided in our paper [1].

Acknowledgments

This work was supported by the European Regional Development Fund under the project AI & Reasoning (reg. no.CZ.02.1.01/0.0/0.0/15_003/0000466).

References

[1] Yutaka Nagashima. LiFtEr: Language to encode induction heuristics for Isabelle/HOL. In *Programming Languages and Systems - 17th Asian Symposium, APLAS 2019, Nusa Dua, Bali, Indonesia, December 1-4, 2019, Proceedings*, pages 266–287, 2019.

[2] Yutaka Nagashima et al. data61/PSL. https://github.com/data61/PSL/releases/tag/v0.1.3-alpha, 2019.

[3] Tobias Nipkow and Gerwin Klein. *Concrete Semantics - With Isabelle/HOL*. Springer, 2014.