A mechanized proof of correctness of the Horn algorithm

Extended Abstract

Marco Giunti *

NOVA LINCS, New University of Lisbon, Portugal

Background This abstract presents a mechanized proof of correctness of a variant [4] of the Horn algorithm in the automated prover Why3 [2]. The paper [4] introduces a functional presentation of the Horn algorithm [3] that is based on a recursive formulation. This is in contrast with the usual pseudo-code imperative presentations of the Horn algorithm, and allows 1) a concise, yet rigorous, formalisation; 2) a clear form of visualising executions of the algorithm, step-by-step; 3) precise results, simple to state and with clean inductive proofs. Roughly, the algorithm receives a conjunction of Horn clauses presented as implications of the form $A \to B$, and recursively accumulates a set of symbols $B_1, \ldots, B_n$ by inspecting if $A_1, \ldots, A_n$ are contained in the dynamically built set. The procedure ends when a fixpoint is reached, that is the next iteration does not change the accumulator, and returns 1 whenever $\bot$ is not in the set, and 0 otherwise. The paper proves that the algorithm is sound and complete: the result is 1 if and only if the formula is satisfiable, and the result is 0 if and only if the formula is contradictory.

Implementation in WhyML *WhyML* is a first-order language with polymorphic types, pattern matching, and inductive predicates. We specified a variant of the algorithm of [4] in WhyML, and verified both tha the algorithm terminates, and that is correct (sound). While the presentation in [4] is fully functional, in our implementation we did mix functional and imperative features. More in details, the algorithm is implemented as a three-layers program. The inner layer is implemented by the recursive function `algR : list boolH -> list boolA -> i -> (list boolH, list boolA)`, where `boolH` is a horn implication, `boolA` is a horn symbol, and `i` is the (ghost) type of propositional variables [1]. The intermediate layer does use an imperative array: `algRA26 : list boolH -> list boolA -> array (list boolH, list boolA) -> i -> (list boolH, list boolA)`. The aim is to track the intermediate results produced by the calls to `algR`, and to enforce invariants that describe the semantics of the algorithm, e.g. the set of horn clauses decreases while the set of horn symbols increase. The top level call is implemented by `alga2: list boolH -> i -> (array (list boolH, list boolA), list boolA)`: the input is the horn formula (described as a conjuction of horn

* Work mainly supported by the Tezos Foudation, via project FACTOR.

[1] The implementation relies on a theory of booleans that is built upon two sub-theories presenting an algebraic and a lattice-based view of booleans, respectively. The theory has beed developed to provide richer semantics to booleans in the proofs.

clauses, hence the list), and the type parameter; the output is a pair of an array (to ensure post-conditions) and of a list of symbols (this is the result in the presentation in [4]).

Mechanized proof The proof of correctness is composed by 75 lemmas and 10 VCs. Most results have been gained by guiding the prover in order to build the proof tree, i.e. à la Coq, which is unusual for the why3 platform (but yet interesting since it shows the capabilities of the framework), and is motivated by both the complexity of the proof (1K loc), and the (better) familiarity of the author with Coq [1]. The key result is Proposition 3.5, which we outline below to provide a flavour of the setting. The lemma says that for all horn formulas f, propositions A, B, and variables' types `i` such that f is both SAT and contains an implication of the form $A \to B$, we have that or B is in the result, or the negation of A is in the result, where the result is `snd (algA2 f def)`, function `castH` casts a symbol of type `boolA` to a (horn) symbol of type `boolH`, function `eval` is the usual evaluation of propositional formulae, function `evalOrList` is a specialized evaluation for disjunctions, and function `or_of_neg_listH2` maps a proposition to a disjunction of negatives.

```
lemma prop35_aux3 : forall f : list boolH, a b : boolH, def : i.
  (horn_formula f /\ noDup f /\
  horn_sat f def /\  mem (ImpH a b) f) ->
  let gen_eval (c : list boolA) : i -> t =
    fun x ->
      if mem (VarA x) c
      then top
      else bot in
  let gc = gen_eval (snd (algA2 f def)) in
  eval (castH b) gc = top \/ evalOrList (or_of_neg_listH2 a) gc = top
```

Discussion We are exploring the feasibility of obtaining a mechanised proof of a fully recursive implementation of the Horn algorithm in Why3. These results, together with the original presentation of the algorithm in [4], are foreseen to be presented in an article to be submitted to a premier venue.

References

1. Coq 8.9.0 – Reference Manual, `https://coq.inria.fr/distrib/current/refman`, accessed May 2019
2. Filliâtre, J.C., Paskevich, A.: Why3 — where programs meet provers. In: ESOP. LNCS, vol. 7792, pp. 125–128. Springer (2013)
3. Horn, A.: On sentences which are true of direct unions of algebras. Journal of Symbolic Logic **16**(1), 14–21 (1951)
4. Ravara, A.: A simple functional presentation and an inductive correctness proof of the Horn algorithm. In: HCVS. EPTCS, vol. 278, pp. 34–48 (2018). https://doi.org/10.4204/EPTCS.278.6