HOL Light QE Update

Johanna Schwartzentruber

Contents

1 Introduction

This document serves as a reference for the major changes and additions to HOL Light QE. Following Patrick's format, the following information will be provided per change, the file it occurs, along with an explanation.

2 fusion changes

2.1 check and Problem

`check` and `Problem` work together to essentially give more informative error messages. For example if an inference rule requires its input(s) to be eval-free, then the error message when one is not eval-free can provide where exactly in the term an evaluation occurs.

2.2 NOT-EFFECTIVE-IN

`not_effectives` is a list of pairs of terms consisting of a variable, v, along with the term it is not-effective-in, t. Pairs can only be added to this list by proving NOT-EFFECTIVE-IN(v, t, u). During variable substitution, when a term has an eval term, in order to simplify substitution this list is searched for the corresponding variable and term. This `not_effectives` list seems to be a spot for improvement as it is quite an inefficient way of storing this information (which must be done in order to do substitution).

2.3 is_eval_free

This function now returns an output of type check, which, if a term is eval-free, will return Ok, otherwise it will return Issue along with the evaluation term.

2.4 is_hole_free

Similar to is_eval_free but for holes.

2.5 Holes

Much of the interactions with holes was changed- most importantly, the type of a hole is now the type of the term represented by the to-be-filled construction and variable substitution inside holes is now (hopefully) correct.

2.6 vars_in

A function to find all the variables in a term (free or not). This is used in tactics when choosing what variables to use in the NEI theorems.

2.7 Inference Rules

A few tweaks were made to the names, also all of the epsilon inference rules now solely deal with the AST of a term, not just any term of type epsilon. For example, quotations may be used as input, but in the final theorem the AST of the quoted term will be used instead.

2.8 construction_type

Returns the term of type ‘type’ that represents the type of the construction. For example: construction_type ‘QuoConst “Zero” (TyBase “nat”)’ returns ‘TyBase “nat”’.

2.9 properConst

Used in constructionToTerm when converting the construction of a constant to the corresponding constant. This is to ensure the constant is an instance of an already defined constant.

2.10 is_construction

Checks if the term is a well-typed, possible construction. Note, it does not necessarily have to be a proper construction (i.e. is the AST of a term).

2.11 proper_e_term

Checks if a term is a proper construction (i.e. is a construction of an already defined term), is a possible proper construction (i.e. a well-typed construction) or is a quoted term. Used to check the inputs of the inference rules.

2.12 orda

A specific case was made explicit (App and Comb) in order to catch when instantiating or substitution requires a NOT-EFFECTIVE-IN theorem to be proved.

2.13 BETA_RED_BY_SUB

Creates the theorem (\x. B) A = SUB(A, x, B).

3 Other file additions and changes

3.1 Constructions\nat_arithmetic.ml

Creation of the types `nat` and `bin`, the representations of unary and binary numbers. Included are the addition and multiplication functions of each type, and proofs that the types are indeed isomorphic.

3.2 holQE.ml

Intended to work as `hol.ml` does- used to load the basics of QE into the already loaded HOL light. As 'kernel' files are added to `Constructions/`, this file should be updated to ensure they are loaded.

3.3 holtest.ml

The kernel along with examples of QE are now tested in holtest. Again, this file should be updated to ensure that when changes are made to system, QE doesn't break.

3.4 Constructions\LEM_hol.ml

The LEM formula with holes, along with an instantiation example. Can be used as reference for how to handle holes.

3.5 Constructions\QE_tests.ml

A test file to test (very) important functions from fusion. Can be added to as one sees fit!

3.6 Constructions\QuotationTactics.ml

This file contains all the general tactics that pertain to terms of type epsilon. All non-trivial tactics (i.e. ones that don't just turn a conversion from fusion to a tactic) should be well documented in the file.

Note: There is still lots of room for improvement with these tactics, they can be made to be more general so they can be applied to more cases, and I am sure they can be made to be more efficient (as some of them do make proofs slow). If\ when they are changed, be sure to check that the `eps_addition` and `eps_multi` files don't break.

3.7 Constructions\gen_tactics.ml

Contains general tactics that could be a part of plain HOL Light. Again, all tactics should be documented in the file.

4 eps_additon.ml

To begin, we are working with the constructions of terms of type nat (which was created in `nat_arithmetic`). We chose to wrap the type in constructor-like functions ('One', 'thenZero', 'thenOne') which allow us to manipulate the terms as if they were binary numbers.

We then must classify our subset of constructions representing proper nat terms. Constructions of nat terms are any (well-formed) terms whose constants only consist of either 'Zero', 'One', 'thenZero', 'thenOne', 'QuoConst', or 'App' with no variables. Since we are working with binary-like numerals, we would like to exclude all binary numerals with leading zeros, thus the purpose of 'start_with_one'. Note, the binary numerals are implemented backwards, so the number two in binary would be implemented as 01, or 'thenZero One'.

Next comes the syntactic algorithm of addition. Note, this algorithm is isomorphic to 'add_bin' in `nat_arithmetic`.

I then began proving 'add_meaning', proving lemmas as I needed them. The lemmas are separated into different groups, those pertaining to type nat and type epsilon. There are also a few nat specific tactics (ex. PROPER_TYPE_TAC) that may be replicated for another subset of constructions, as long as the corresponding lemmas and theorems they reference are also replicated. An example of this was done in the `eps_multi` file which is explained below.

After completing the proof, in order to shorten it, I noticed many theorems (especially for the NOT-EFFECTIVE-IN (NEI) lemmas) were quite similar, so I created a few 'theorem templates' that take some arguments and return the corresponding theorem. Next came the NEI lemmas which were quite repetitive (and not well named), but they were proved, again, as I needed them. Something important I feel I should point out is that for the second induction I required the induction theorem to have different instantiated variables. For this reason I created 'eps_alt_ind' and instantiated that as the second induction formula. This problem will probably occur again, that is a get around.

Finally, lemmas were created for specific cases in the induction proof to make it more readable (though it is still pretty unreadable).

5 eps_multi.ml

The same approach as above was taken in proving the multiplication meaning formula. What was easier was that the 6 lemmas pertaining to the subset of proper nat constuctions (mult_lemma#) were almost identical to the 6 appearing in the `eps_addition` file. This leads me to believe that tactics can be made for these, and since they aren't nat-specific, they may be able to be generalized to other subsets of constructions.

Overall, there are many similarities between the proofs, besides the 6 lemmas. Both used similar theorem templates and NEI lemmas. Tactics?