CS101C Homework 8

Due: Wednesday, May 28, 9PM (firm)

Collaboration: You are allowed and encouraged to work together and collaborate on all aspects of this homework. However, your submission **must** be your own; you must type in your homework without referring to shared or other "external" material. For example, suppose you work as part of a group to prove a long, complicated theorem; and suppose you sketch the proof on the board. When you enter the proof into MetaPRL for homework submission, you must not refer to the board — you must recover the proof from your own memory.

Setup

Start this homework by updating MetaPRL to revision **11** (e.g. version "`0.8.3 (CS101 rev 11)`"). Upgrade instructions are available at `http://nogin.org/cs101c/mp-update.html`.

In this homework you will be extending the `cs101_hw8` theory (which is a simplified version of the theory of collection type, `itt_collection`). In directory `theories/itt` of your MetaPRL installation, rename the files

`cs101_hw8.ml`		`cs101_hw8_`*name*`.ml`
`cs101_hw8.mli`	into	`cs101_hw8_`*name*`.mli` respectively
`cs101_hw8.prla`		`cs101_hw8_`*name*`.prla`

(where *name* is your login name). In file `cs101_hw8_`*name*`.prla`, replace all occurrences of string `Cs101_hw8` with `Cs101_hw8_`*name* (use your favorite text editor's "replace all" functionality). Finally, add `cs101_hw8_`*name* to the end of the `MPFILES` variable in the `theories/int/Makefile`.

For this homework, you should be working within the `ITT` theory. You are not allowed to add any new `prim` rules or rewrites to the system and you are not allowed to modify the system in any way, other than extending it with your new `hw8` module.

Note: after you change the `MPFILES` variable in the `Makefile` or add a new `extends` or `open` directives to a MetaPRL file and before you run `make opt`, you might need to run `make depend` to update the cross-module dependencies.

In all problems of this homework you may add extra well-formedness assumption(s) when it is necessary to make the rules provable. Make sure you only add the one(s) that are truly necessary. In particular, do *not* add

well-formedness assumptions for variables that already have one. Also, you might find it useful to formulate and prove some "intermediate" lemmas, maybe even define new operator(s).

This homework requires good understanding of quotient and esquash type operators. For more information on those operators, see the `theories.pdf` and the "Quotient Types: A Modular Approach" paper. The "quotient" paper also has some information on the type of collections that might be useful for the part II of this homework.

Part I: Unordered pairs

1. Define a `pairs`$\{T\}$ operator such that whenever T is a type, then `pairs`$\{T\}$ is a type of *unordered* pairs of elements of T. In other words, two terms should be equal in `pairs`$\{T\}$ if and only if each one is a pair of elements of T and the seconds ones contains the same elements of T as the first one, possibly in the reverse order.

   Prove that `pairs`$\{T\}$ is a type whenever T is a type and that `pairs`$\{T\}$ is in $\mathbb{U}_i$ whenever T is in $\mathbb{U}_i$.

   Optional: to make sure your definition is correct, prove $(1,1) \in$ `pairs`$\{\mathbb{Z}\}$ ($\mathbb{Z}$ is a type of integers, `Itt_int_base!int`, use `int` to type it in) and $(2,1) = (1,2) \in$ `pairs`$\{\mathbb{Z}\}$

2. Define a `sum`$\{p\}$ operator that given a pair of integers returns the sum of the two components of the pair. Prove that `sum`$\{\mathrm{p}\} \in \mathbb{Z}$ whenever $p \in$ `pairs`$\{\mathbb{Z}\}$

   Optional: to make sure your definition is correct, prove `sum`$\{(1,2)\} \leftrightarrow 3$.

   Hints: Integer addition operation is `Itt_int_base!add`; use $a +@ b$ to type it in. You can use `add_Commut` rule to prove commpuativity of addition and/or `add_CommutC` conversion (you might have to call it through `addrC` to reorder arguments) in an `add` term.

Part II: Lists

1. Define a `col_sub`$\{c_1; c_2\}$ operator such that `col_sub`$\{c_1; c_2\}$ is a collection that contains all elements of collection c_1 that are not members of collection c_2. There may be many different ways of defining `col_sub`, make sure that you pick a definition that would make all the statements below provable.

2. Prove that whenever c_1 and c_2 are members of $\texttt{Col}_i\{T\}$, then $\texttt{col\_sub}\{c_1; c_2\}$ is also a member of that type.

3. Prove the rules:

$$\frac{\Gamma \vdash t \in_c c_1 \qquad \Gamma \vdash \neg(t \in_c c_2)}{\Gamma \vdash t \in_c \texttt{col\_sub}\{c_1; c_2\}}$$

$$\frac{\Gamma \vdash t \in_c \texttt{col\_sub}\{c_1; c_2\}}{\Gamma \vdash t \in_c c_1}$$

$$\frac{\Gamma \vdash t \in_c \texttt{col\_sub}\{c_1; c_2\}}{\Gamma \vdash \neg(t \in_c c_2)}$$

where $\in_c$ (displayed $\in_{(c)}$ in `xterm`) is the collection `mem` operator.

Submission Instructions

Make sure you `export` all the proofs. Send all the files `cs101_hw8_`*name*`.ml`, `cs101_hw8_`*name*`.mli` and `cs101_hw8_`*name*`.prla` as text attachments in an email to `cs101-admin@metaprl.org`. Please include "`CS101 HW8`" in the message subject line.