CS101C Homework 9

Due: Wednesday, June 4, 11:59PM (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 **12** (e.g. version "`0.8.3 (CS101 rev 12)`"). Note: when upgrading between revisions **11** (or earlier) and **12** (or later), there is an extra step in upgrade process. Upgrade instructions are available at `http://nogin.org/cs101c/mp-update.html`.

In directory `theories/itt` of your MetaPRL installation, create files `cs101_hw9_`*name*`.ml` and `cs101_hw9_`*name*`.mli` (where *name* is your login name). Add `cs101_hw9_`*name* to the end of the `MPFILES` variable in the `theories/itt/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 `hw9` 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).

Part I: Top

Prove that the types $Void \to A$ and $A \to$ Top are extensionally equal. *Hints*: the extensional equality operator, `ext_equal` is defined in the `Itt_ext_equal` module; use `fnExtenT` tactic (documented in `itt_quickref.txt`).

Part II: Records

Hint: You might want to take a look at the `Itt_record_exm` module. It should give you a better idea of how to type in record types and records and how to prove their properties.

1. Using the record type constructors defined in `Itt_record`, define the type constructor `stacks` such that for any type T, $\texttt{stacks}[i\colon \texttt{l}]\{T\}$ is a type of all possible records that have the following fields:

   (a) `t` of type $\mathbb{U}_i$,

   (b) `empty` of type `t`,

   (c) `is_empty` of type $\texttt{t} \to \texttt{Bool}$

   (d) `push` of type $\texttt{t} \to T \to \texttt{t}$,

   (e) `pop` of type $\{s : \texttt{t} \mid \neg(\uparrow (\texttt{is\_empty}\, s))\} \to (T \times \texttt{t})$.

   Prove that for an appropriate T, $\texttt{stacks}[i\colon \texttt{l}]\{T\} \in \mathbb{U}_{i'}$ and $\texttt{stacks}[i\colon \texttt{l}]\{T\}$ is a type.

2. What properties of `t`, `empty`, `is_empty`, `push` and `pop` would you expect a member of the $\texttt{stacks}[i\colon \texttt{l}]\{T\}$ type to have, if it really implements a stack, and not something random? Define a predicate $\texttt{is\_valid}\{T; s\}$ that given $T \in \mathbb{U}_i$ and $s \in \texttt{stacks}[i\colon \texttt{l}]\{T\}$ states that s is a "correct" implementation. (*Hint*: The definition I would give would have 3 separate clauses joined by "and"s.) Define

   $$\texttt{valid\_stacks}[i\colon \texttt{l}]\{T\} := \{s : (\texttt{stacks}[i\colon \texttt{l}]\{T\}) \mid \texttt{is\_valid}\{T; s\}\}$$

   and prove that $\texttt{valid\_stacks}[i\colon \texttt{l}]\{T\}$ is a type.

3. Using any ITT operators you want, define a `stack` operator such that for any $T \in \mathbb{U}_i$, $\texttt{stack}\{T\} \in \texttt{valid\_stacks}[i\colon \texttt{l}]\{T\}$. (*Hint*: forget about MetaPRL and ITT for a second and think how you would implement this in OCaml if you could not use mutable data structures. Then do the same thing in ITT.) Prove that $\texttt{stack}\{T\}$ indeed has the given type.

Part III (optional): Course feedback

This part is optional. If you want, you can submit your comments anonymously (see the "Submission Instructions" section below).

Please let us know what you thought of this course. Were the lectures easy to understand? Were they too fast/too slow? Were the homeworks too hard/too easy? Any suggestions on how this course could be improved? How hard was MetaPRL to learn and use? Is there any functionality that you think MetaPRL is missing?

Thanks a lot for any feedback you can provide!

Submission Instructions

Parts I and II: Make sure you `export` all the proofs. Send all the files `cs101_hw9_`*name*`.ml`, `cs101_hw9_`*name*`.mli` and `cs101_hw9_`*name*`.prla` as text attachments in an email to `cs101-admin@metaprl.org`. Please include "`CS101 HW9`" in the message subject line.

Part III: You can email your feedback to `cs101-admin@metaprl.org` (please include "`CS101 feedback`" in the message subject line, if possible). If you want to remain anonymous, feel free to use an anonymizing remailer (such as, for example, `https://riot.eu.org/anon/remailer.html.en`; or see a list at `http://dmoz.org/Computers/Internet/E-mail/Anonymous_Mailers/`) or leave it in Aleksey Nogin's or Xin Yu's mailbox on 2nd floor in Jorgensen.