Constructive Logic (15-317), Fall 2012
Assignment Twelf: Bracket Abstraction Metatheory

Carlo Angiuli (`cangiuli@cs`)

Out: Thursday, November 29, 2012
Due: Friday, December 7, 2012 (5:00 pm)

With the full power of higher-order encodings in your toolbelt, you're ready to use Twelf for metatheory. In this **extra credit** assignment, you'll prove correctness of your encoding of bracket abstraction from Assignment 10. You should refer to your filled-in `bracket.elf` and add your code to the newly-released `bracket.thm`.

There are no written portions of this assignment; just copy your completed `bracket.thm` file to

```
/afs/andrew/course/15/317/submit/<userid>/hwTwelf
```

where `<userid>` is replaced with your Andrew ID. Your solutions should work in the version of Twelf installed in the course directory.

1 Warmup: Worlds and Totality

One simple kind of correctness can be checked without doing metatheory proper: the effectiveness of the translation. To do this, you'll have to think carefully about the *worlds* in which translation is total.

Extra Credit Task 1 (10 points)**.** Add `%mode`, `%worlds`, and `%total` declarations to bracket and translate.

2 Proof Development Tips

In the next section, you'll prove a Twelf metatheorem. Along the way, you may find yourself at a loss for what to do—here are some base guidelines to avoid that situation.

- Do the proof (or a tricky case of the proof) on paper before attempting it in Twelf. This will help you discover whether your confusion is due to the proof itself or its Twelf encoding.

- If you want to work on a part of the proof that uses a lemma before you prove the lemma, use `%trustme`. To use it, type `set unsafe true` at the Twelf top level, and write `%trustme` before any directive in your file. Remember to `set unsafe false` and test your code a final time before you hand it in.
- Write type annotations on all inputs to a clause of the theorem, and on all outputs of subgoals (i.e., inductive calls or lemma calls).
- Refer frequently to the output of type reconstruction. If you're not sure how to give a type annotation as suggested above, see how much you can get Twelf to tell you first.
- Start by writing underscores (or conspicuous metavariables like `XXX`) in the output positions of the case, do some work on the subgoal side (such as applying induction), and see how close the outputs of your subgoals get you to the reconstructed output type.
- If all else fails, return to the paper proof. Try naming the derivations on paper: it will get you closer to Twelf notation.

3 Proving Dynamic Correctness

The simply-typed lambda calculus and the combinator calculus each have static and dynamic semantics that define what the systems mean and how you compute with them. What you are going to prove is that your definition of the translation from one to the other is correct with respect to those semantics. In English, the theorem you'll prove is that if a lambda term e takes a step to e', and if e translates to the combinator term C, then e translates to some combinator term C' to which C reduces.

The notion of *reduces* we need here is slightly different from the *step* relation we defined on combinators before. It differs for one in that an application can reduce its second argument, and for another in that it is closed under reflexivity and transitivity. Its definition is provided.

```
creduce : comb -> comb -> type.
creduce/S : creduce (capp (capp (capp cS A) B) C)
                    (capp (capp A C) (capp B C)).
creduce/K : creduce (capp (capp cK A) B) A.
creduce/I : creduce (capp cI A) A.
creduce/app1 : creduce (capp A B) (capp A' B)
                <- creduce A A'.
creduce/app2 : creduce (capp A B) (capp A B')
                <- creduce B B'.
creduce/trans : creduce C C''
                <- creduce C C'
                <- creduce C' C''.
creduce/refl : creduce C C.
```

On paper, we'll write `creduce C C'` as $C \gg C'$.

The first thing we need to prove on the way to proving translation correct is a lemma about bracket abstraction. We need to describe the sort of computational behavior we expect from bracket abstracted terms. Since they are, in essence, functions, they should have a β-like behavior:

Lemma 1 (Combinator Substitution)**.** If $\langle x\rangle C1 = C1'$, then for any combinator term $C2$, $C1'@C2 \gg [C2/x]C1$.

Extra Credit Task 2 (10 points)**.** Prove the combinator substitution lemma in Twelf by filling in clauses for the following declaration.

```
csubst : bracket ([x] C x) C* -> {C':comb}
             creduce (capp C* C') (C C') -> type.
%mode csubst +X1 +C' -X2.

%% Fill in cases here

%worlds () (csubst _ _ _).
%total D (csubst D _ _).
```

Theorem 1 (Dynamic Correctness)**.** If $E \mapsto E'$ and $\text{tr}(E) = C$ then $\text{tr}(E') = C'$ and $C \gg C'$.

Extra Credit Task 3 (10 points)**.** Prove the dynamic correctness theorem in Twelf by filling in clauses for the following declaration.

```
dyn-thm : step E E' -> translate E C
           -> translate E' C'
           -> creduce C C' -> type.
%mode dyn-thm +X1 +X2 -X3 -X4.

%% Fill in cases here

%worlds () (dyn-thm _ _ _ _).
%total D (dyn-thm D _ _ _).
```

4 Proving Static Correctness

Static correctness, the idea that translation preserves types, is notionally a simpler theorem, but it requires more subtle Twelf machinery. Therefore, this problem is *extra* extra credit—only do it if you're having fun and want to learn more.

The statement of the theorem is as follows:

```
stat-thm : translate E C -> of E T -> cof C T -> type.
%mode stat-thm +X1 +X2 -X3.
%block tbind : some {T:tp}
  block {x:term} {dx:of x T}
        {cx:comb} {dcx:cof cx T}
        {dtrans:translate x cx}
        {dthm : stat-thm dtrans dx dcx}.
%worlds (tbind) (stat-thm _ _ _).
%total D (stat-thm _ D _).
```

You'll need to prove a corresponding lemma about bracket abstraction:

```
brack-stat-thm : ({x} cof x T -> cof (C x) T') -> bracket ([x] C x) C*
                 -> cof C* (arrow T T') -> type.
%mode brack-stat-thm +X1 +X2 -X3.
%worlds (tbind) (brack-stat-thm _ _ _).
%total D (brack-stat-thm _ D _).
```

Extra Credit Task 4 (10 points)**.** Prove the static correctness of bracket abstraction lemma in Twelf by filling in clauses for `brack-stat-thm`.

And finally, you'll need a way to prove that if you have a derivation about a combinator term that doesn't have a free variable, the derivation itself lacks the variable. This proof is provided.

```
cof-strengthen : ({x} {d:cof x T} cof C T')
                  -> cof C T' -> type.
%mode cof-strengthen +X1 -X2.
-D : cof-strengthen ([x] [d] D) D.
-app : cof-strengthen ([x] [d] cof/app (Dof2 x d) (Dof1 x d))
           (cof/app Dof2* Dof1*)
        <- cof-strengthen Dof2 Dof2*
        <- cof-strengthen Dof1 Dof1*.
%worlds (tbind) (cof-strengthen _ _).
%total D (cof-strengthen D _).
```

Extra Credit Task 5 (10 points)**.** Prove the static correctness theorem in Twelf by filling in clauses for `stat-thm`.