Constructive Logic (15-317), Spring 2023
Assignment 5: Continuations and Arithmetic
(60 points)

Instructor: Frank Pfenning

Due: March 21, 2023, 11:59 pm

This assignment will have a written portion, a programming portion and a Dcheck portion. You will submit all portions through Gradescope.

We recommend that you typeset your written solutions. Most students use LaTeX, but other software is acceptable. If you choose not to typeset your solutions, be aware that your handwriting must be **legible**.

1 Classical Sequent Calculus (3 points)

In class, we introduced classical sequent calculus. Below is an example proof of $\cdot \overset{\mathsf{CL}}{\Longrightarrow} (\neg\neg A) \supset A$. In the classical sequent calculus, we may omit both antecedents and succedents that are no longer needed (silently applying the admissible rule of weakening either on the left or right).

$$\dfrac{\dfrac{\dfrac{\dots\dots\ \mathsf{id}}{A \overset{\mathsf{CL}}{\Longrightarrow} \bot, A}\supset R \qquad \dfrac{}{\bot \overset{\mathsf{CL}}{\Longrightarrow} A}\bot L}{\cdot \overset{\mathsf{CL}}{\Longrightarrow} \neg A, A}\supset L}{\dfrac{\neg\neg A \overset{\mathsf{CL}}{\Longrightarrow} A}{\cdot \overset{\mathsf{CL}}{\Longrightarrow} (\neg\neg A) \supset A}\supset R}$$

Task 1. (3 points) Prove the following sequent using classical sequent calculus. For reference, you may find the inference rules for classical sequent

in Appendix A.

$$\cdot \stackrel{\mathsf{CL}}{\Longrightarrow} ((A \supset B) \supset A) \supset A$$

(no **Task 2**)

2 Classical Proof Terms (10 points)

Using the proof terms for classical logic in Lecture 12, write out proof terms for the following two propositions. Note: $\neg A$ is the type of continuations expecting to be thrown a value of type A, rather than the intuitionistic abbreviation $A \supset \bot$.

Task 3. (5 points)

$$\neg(A \wedge B) \supset (\neg A \vee \neg B)$$

Task 4. (5 points)

$$(A \supset B) \supset (\neg A \vee B)$$

3 Computing with Continuations (8 points)

Examples, typing and computation rules for continuation are given in Lecture 12 and in B.

Task 5. (8 points) Consider the following proof terms.

$$M \triangleq \mathbf{callcc}(k.\, \mathbf{inl}\ (\mathbf{callcc}(k'.\, \mathbf{throw}\ k\ (\mathbf{inr}\ k'))))$$
$$N \triangleq \mathbf{case}(M, x.\, x, y.\, \mathbf{throw}\ y\ 317)$$

Let $\mathcal{C}$ be some arbitrary context. Evaluate $\mathcal{C}[N]$ until you find some $\mathcal{C}[P]$ with P *value* (where you should assume that 317 is a value). Do not skip any of the reduction steps.

4 Tail recursion (17 points)

You can find relevant signature in the Lecture 12 notes and also in the Lecture 12 code cont.sml and tcheck-cont.sml.

In the lecture, we have seen examples that how we can use continuations mostly in the style of exceptions. In this question, we will explore tail recursion using continuation.

As we have learned in 15-150, if we have pending computations that make use of the result of the recursive call, the stack usage is going to rise. Functions that do not have pending computations are said to be tail recursive, and the recursive call is called a tail call.

Consider the following recursive function.

```
datatype nat = Z | S of nat
fun dbl (n : nat) : nat =
  case n of
    Z => Z
  | S n' => S (S (dbl n'))
```

Clearly this function is not tail recursive, since the second case has further computations that depend on the result of the recursive call.

Task 6. (6 points) Implement a function `dblcc` that has type `nat * nat cont -> void`. In addition of the original argument `n`, the new function takes in a continuation that has type `nat cont`. Rather than return a value, the function throws that value to the given return point and has return type `void`, which means it never returns.

```
datatype void = Void of void
fun abort (Void _) = raise Fail "impossible"

fun dblcc (n : nat, k : nat cont) : void =
  (* Your implementation here *)
```

Constraints: You should use `callcc` at most once. Your function `dblcc` does not need to be tail recursive.

Task 7. (3 points) Implement a function `dbl'` that has type `nat -> nat` using `dblcc` you wrote in Task 6 by providing appropriate numeric and continuation arguments. The result of `dbl'` should be exactly as same as `dbl`.

```
fun dbl' (n : nat) : nat =
  (* Your implementation here *)
```

Task 8. (8 points) Based on `dblcc` you wrote, implement a tail recursive function `dblcps` that has type `nat * nat cont -> void`. The behavior of `dblcps` should be as same as `dblcc`.

```
fun dblcps (n : nat, k : nat cont) : void =
  (* Your implementation here *)
```

Constraints: Your `dblcps` should be tail recursive, meaning that no further computations should be made after the recursive call `dblcps`.

Hint: Directly calling `dblcc` will not make your code tail recursive, since the function `dblcc` does not achieve tail recursion.

Task 9. (0 points) Implement a function `dbl''` by replacing the usage of `dblcc` by `dblcps` in `dbl'`. You should not write anything new in this task, since we just want you to test your implementation of `dblcps`.

```
fun dbl'' (n : nat) : nat =
  (* Copy your dbl' code and change dblcc to dblcps *)
```

5 Predicate Calculus (10 points)

Using Dcheck, give derivations of the following judgements, if they are derivable. For the ones that are not derivable, simply put:

```
deriv <name> = omitted
```

Note: Although this problem does not involve natural numbers, we are working in the Heyting Arithmetic (AR) system because that's where quantifiers are available in Dcheck. This means you need to write the quantifiers $\forall x.\, A(x)$ as `All x:T. A(x)` and $\exists x.\, A(x)$ as `Exists x:T. A(x)` where `T` represents an arbitrary (unspecified) domain of quantification in order to distinguish it from nat.

Task 10. (5 points)

$$(\forall x.\, A(x)) \supset (\exists x\, .A(x))\ \mathit{true}$$

Task 11. (5 points)

$$((\forall x.\, A(x)) \wedge (\exists x.\, \top)) \supset (\exists x.\, A(x))\ \mathit{true}$$

6 Arithmetic (12 points)

Task 12. (12 points) Using Dcheck, give derivation for the following judgment.

$$\forall x : \mathsf{nat}.\, \forall y : \mathsf{nat}.\, \forall z : \mathsf{nat}.\, (x = y) \supset (y = z) \supset (x = z)\ \mathit{true}$$

Hint: This problem is challenging. One suggestion is not to introduce all three quantifiers at once since the resulting induction hypothesis may be too weak.

A Classical Sequent Calculus

$$\frac{}{\Gamma, P \overset{\mathsf{CL}}{\Longrightarrow} P, \Delta}\ \mathsf{id}^*$$

$$\frac{}{\Gamma, \bot \overset{\mathsf{CL}}{\Longrightarrow} C, \Delta}\ \bot L \qquad \frac{}{\Gamma \overset{\mathsf{CL}}{\Longrightarrow} \top, \Delta}\ \top R$$

$$\frac{\Gamma \overset{\mathsf{CL}}{\Longrightarrow} A, \Delta \quad \Gamma \overset{\mathsf{CL}}{\Longrightarrow} B, \Delta}{\Gamma \overset{\mathsf{CL}}{\Longrightarrow} A \wedge B, \Delta}\ \wedge R \qquad \frac{\Gamma, A, B \overset{\mathsf{CL}}{\Longrightarrow} C, \Delta}{\Gamma, A \wedge B \overset{\mathsf{CL}}{\Longrightarrow} C, \Delta}\ \wedge L$$

$$\frac{\Gamma \overset{\mathsf{CL}}{\Longrightarrow} A, B, \Delta}{\Gamma \overset{\mathsf{CL}}{\Longrightarrow} A \vee B, \Delta}\ \vee R \qquad \frac{\Gamma, A \overset{\mathsf{CL}}{\Longrightarrow} \Delta \quad \Gamma, B \overset{\mathsf{CL}}{\Longrightarrow} \Delta}{\Gamma, A \vee B \overset{\mathsf{CL}}{\Longrightarrow} \Delta}\ \vee L$$

$$\frac{\Gamma, A \overset{\mathsf{CL}}{\Longrightarrow} B, \Delta}{\Gamma \overset{\mathsf{CL}}{\Longrightarrow} A \supset B, \Delta}\ \supset R \qquad \frac{\Gamma \overset{\mathsf{CL}}{\Longrightarrow} A, \Delta \quad \Gamma, B \overset{\mathsf{CL}}{\Longrightarrow} \Delta}{\Gamma, A \supset B \overset{\mathsf{CL}}{\Longrightarrow} \Delta}\ \supset L$$

$$\frac{}{\Gamma, A \overset{\mathsf{CL}}{\Longrightarrow} A, \Delta}\ \mathsf{id} \qquad \frac{\Gamma \overset{\mathsf{CL}}{\Longrightarrow} A, \Delta \quad \Gamma, A \overset{\mathsf{CL}}{\Longrightarrow} \Delta}{\Gamma \overset{\mathsf{CL}}{\Longrightarrow} \Delta}\ \mathsf{cut}$$

$$\frac{\Gamma \overset{\mathsf{CL}}{\Longrightarrow} \Delta}{\Gamma \overset{\mathsf{CL}}{\Longrightarrow} A, \Delta}\ \mathsf{weaken}R \qquad \frac{\Gamma \overset{\mathsf{CL}}{\Longrightarrow} \Delta}{\Gamma, A \overset{\mathsf{CL}}{\Longrightarrow} \Delta}\ \mathsf{weaken}L$$

$$\frac{\Gamma \overset{\mathsf{CL}}{\Longrightarrow} A, A, \Delta}{\Gamma \overset{\mathsf{CL}}{\Longrightarrow} A, \Delta}\ \mathsf{contract}R \qquad \frac{\Gamma, A, A \overset{\mathsf{CL}}{\Longrightarrow} \Delta}{\Gamma, A \overset{\mathsf{CL}}{\Longrightarrow} \Delta}\ \mathsf{contract}L$$

B Continuations

We use V for terms M with M *value*.

Local reductions.

$\mathbf{fst}\,\langle M, N\rangle$	$\Longrightarrow_R$	M	$(A \wedge B)$
$\mathbf{snd}\,\langle M, N\rangle$	$\Longrightarrow_R$	N	$(A \wedge B)$
$(\lambda x.\, M)\, V$	$\Longrightarrow_R$	$[V/x]M$	$(A \supset B)$
$\mathbf{case}(\mathbf{inl}\, V, x.\, N, y.\, P)$	$\Longrightarrow_R$	$[V/x]N$	$(A \vee B)$
$\mathbf{case}(\mathbf{inr}\, V, x.\, N, y.\, P)$	$\Longrightarrow_R$	$[V/y]P$	$(A \vee B)$
no local reduction for $\langle\,\rangle$			$(\top)$
no local reduction for $\mathbf{abort}\, M$			$(\bot)$

Evaluation contexts.

C	$::=$	$[\,]$	(general)
	$\mid$	$\mathbf{fst}\, C \mid \mathbf{snd}\, C$	$(A \wedge B)$
	$\mid$	$C\, N \mid V\, C$	$(A \supset B)$
	$\mid$	$\mathbf{inl}\, C \mid \mathbf{inr}\, C \mid \mathbf{case}(C, u.N, w.P)$	$(A \vee B)$
	$\mid$	*(none)*	$(\top)$
	$\mid$	$\mathbf{abort}\, C$	$(\bot)$
	$\mid$	$\mathbf{throw}\, C\, N \mid \mathbf{throw}\, V\, C$	$(\neg A)$

Computation steps.

$$\frac{M \Longrightarrow_R M'}{C[M] \longrightarrow C[M']}$$

$$\frac{}{C[\,]\ \mathit{value}}$$

$$\frac{}{C[\mathbf{callcc}\,(k.\, M)] \longrightarrow C[[C[\,]/k]M]} \qquad \frac{V\ \mathit{value}}{C[\mathbf{throw}\, C'[\,]\, V] \longrightarrow C'[V]}$$