# Midterm II Sample Questions

15-317 Constructive Logic
Frank Pfenning

November 7, 2017

Name: **Sample Solution** Andrew ID: **fp**

## Instructions

- This exam is closed book, closed notes.
- You have 80 minutes to complete the exam.
- There are only 4 problems!

| | Admissibility of Cut | Prolog | Logic Programming | Deriving Rules | | |
|---|---|---|---|---|---|---|
| | Prob 1 | Prob 2 | Prob 3 | Prob 4 | Prob 5 | Total |
| Score | **30** | **30** | **30** | **30** | | **120** |
| Max | 30 | 30 | 30 | 30 | | 120 |

# 1 Admissibility of Cut (30 pts)

In functional languages, we have parametric polymorphism and abstract types, which corresponds to universal and existential quantification over types, respectively. The corresponding constructs in logic are universal and existential quantification over propositions.

In this problem we consider the existential quantifier, given by the following rules in the sequent calculus.

$$\frac{\Gamma \Longrightarrow [B/x]A}{\Gamma \Longrightarrow \exists x{:}\mathsf{prop}.\ A}\ \exists R \qquad \frac{\Gamma, \exists x{:}\mathsf{prop}.\ A, [p/x]A \Longrightarrow C}{\Gamma, \exists x{:}\mathsf{prop}.\ A \Longrightarrow C}\ \exists L^p$$

In the $\exists R$ rule, $B$ must be a proposition. In the $\exists L$ rule, $p$ must be a fresh propositional variable. The *substitution principle for propositions* says that we can always substitute a proposition for a propositional variable throughout a proof; you will need that below.

We state the admissibility of cut as

$$\frac{\overset{\mathcal{D}}{\Gamma \Longrightarrow A} \quad \overset{\mathcal{E}}{\Gamma, A \Longrightarrow C}}{\Gamma \Longrightarrow C}\ \mathsf{cut}_A$$

**Task 1** (20 pts). Show what would be the critical case in the proof of the admissibility of cut, when the cut formula is inferred by $\exists R$ in $\mathcal{D}$ and by $\exists L$ in $\mathcal{E}$. You should only be concerned about eliminating the cut at proposition $\exists x{:}\mathsf{prop}.\ A$; do not be concerned about whether you are actually justified in applying the induction hypothesis (see Task 3).

**Case:** $\mathcal{D} = \dfrac{\overset{\mathcal{D}'}{\Gamma \Longrightarrow [B/x]A}}{\Gamma \Longrightarrow \exists x{:}\mathsf{prop}.\ A}\ \exists R$ and $\mathcal{E} = \dfrac{\overset{\mathcal{E}'}{\Gamma, \exists x{:}\mathsf{prop}.\ A, [p/x]A \Longrightarrow C}}{\Gamma, \exists x{:}\mathsf{prop}.\ A \Longrightarrow C}\ \exists L^p$

$\Gamma, [p/x]A \Longrightarrow C$ (by cut on $\exists x{:}\mathsf{prop}.\ A$, $\mathcal{D}$, and $\mathcal{E}'$) — $\mathcal{F}'$

$\Gamma, [B/x]A \Longrightarrow C$ (by substitution property for propositions and $[B/p][p/x]A = [B/x]A$) — $[B/p]\mathcal{F}'$

$\Gamma \Longrightarrow C$ (by cut on $[B/x]A$, $\mathcal{D}'$, and $[B/p]\mathcal{F}'$) — $\mathcal{F}$

**Task 2** (5 pts). The admissibility of cut is shown by a nested induction, which can also be formulated as a well-founded induction over a lexicographic order. Complete the following paragraph, using $\Gamma$, $A$, $C$, $\mathcal{D}$, and $\mathcal{E}$ from the statement of the admissibility of cut.

The proof is by nested induction, first over [ $A$ ], then over [ $\mathcal{D}$ ] and [ $\mathcal{E}$ ]. This means in an appeal to the induction hypothesis, either [ $A$ ] becomes smaller and [ $\mathcal{D}$ ] and [ $\mathcal{E}$ ] are arbitrary, or [ $A$ ] remains the same and one of [ $\mathcal{D}$ ] and [ $\mathcal{E}$ ] becomes smaller while the other remains the same.

**Task 3** (5 pts). When adding quantifiers over propositions, this particular induction argument fails. Explain why by completing the following paragraph.

The case for $\exists R$ against $\exists L$ would require us to appeal to the induction hypothesis on $[B/x]A$, $\mathcal{D}'$, and $[B/p]\mathcal{F}'$. However, this is *not* smaller than $\exists x{:}\mathsf{prop}.\ A$, $\mathcal{D}$, and $\mathcal{E}$ because $[B/x]A$ may be bigger than $\exists x{:}\mathsf{prop}.\ A$.

## 2 Prolog (30 pts)

One issue with lists in functional programs is that we can access them only at one end. In logic programming, so-called *difference lists* allows us to access both ends! Recall that Prolog notation for a list is `[]` for the empty list, and `[X|Xs]` for the list with head `X` and tail `Xs`. We abbreviate `[X1|[X2|[X3|...|Xs]]]` as `[X1,X2,X3,...|Xs]`.

A difference list with elements $x_1, \ldots, x_n$ has the form `diff([x1,...,xn|L],L)` where `L` is a logic variable, subject to unification. Note that it must be the *same* logic variable `L` at the tail end of the list, and as the second argument to the constructor `diff`, but that different difference lists may have different variables (ha!).

Complete the following programs, always assuming that the given difference lists are in the form shown above.

**Task 1** (5 pts). If `to_list(diff(K,L), Xs)` and `diff(K,L)` represents `[x1,...,xn]` then the ordinary list `Xs` is `[x1,...,xn]`.

```
to_list(diff(K,L), K) :- L = [].
% or
% to_list(diff(K,[]), K).
```

**Task 2** (5 pts). If `prepend(Y, diff(K,L), D)` and `diff(K,L)` represents `[x1,...,xn]`, then the difference list `D` represents `[Y,x1,...,xn]`.

```
prepend(Y, diff(K,L), diff([Y|K], L)).
```

**Task 3** (10 pts). If `postpend(diff(K,L), Y, D)` and `diff(K,L)` represents `[x1,...,xn]`, then the difference list `D` represents `[x1,...,xn,Y]`.

```
postpend(diff(K,L), Y, diff(K,M)) :- L = [Y|M].
% or
% postpend(diff(K,[Y|M]), Y, diff(K,M)).
```

**Task 4** (10 pts). If `concat(diff(K,L), diff(M,N), D)` and `diff(K,L)` represents `[x1,...,xi]` and `diff(M,N)` represents `[y1,...,yj]` then difference list `D` represents `[x1,...,xi,y1,...,yj]`.

`concat(diff(K,L), diff(L,M), diff(K,M))`.

## 3 Logic Programming (30 pts)

We explore some programming over heap-ordered binary trees. A binary tree is *heap-ordered* if the key at every node is less than or equal to the key at its children. We assume we have predicate `N <= K` which succeeds if $N \leq K$. Both `N` and `K` must be ground, that is, free of variables subject to unification. We represent binary trees with `leaf` and `node` constructors.

```
tree(leaf).
tree(node(Left, N, Right)) :- tree(Left), nat(N), tree(Right).
```

**Task 1** (15 pts). Define a *backward chaining* predicate `ordered(Tree)` in Prolog that tests if a given tree is heap-ordered. It should have mode `ordered(+tree)` and always terminate in this mode. You may define auxiliary predicates as needed and use `N <= K`.

```
ordered(leaf).
ordered(node(Left, N, Right)) :-
    below(N, Left), below(N, Right).

below(N, leaf).
below(N, node(Left, K, Right)) :-
    N <= K, ordered(node(Left, K, Right)).
```

**Task 2** (15 pts). Now define a *forward chaining* predicate `ord(T)` that tests if a given tree is heap-ordered. The assertion `ord(T)` for a given (ground) tree $T$ should derive `no` iff the assumption that $T$ is heap-ordered is inconsistent. You may generate additional assertions `leq(N,K)`, assuming that this will prove `no` iff `N` is not less than or equal to `K`. Your program must saturate.

You may give your answer in the form of inference rules (read from the premises to the conclusion), propositions with all positive atoms, or in Prolog syntax (but keeping in mind the clauses will be applied with forward chaining).

$$\frac{\mathsf{ord}(\mathsf{node}(\mathit{Left}, N, \mathit{Right}))}{\mathsf{below}(N, \mathit{Left})} \qquad \frac{\mathsf{ord}(\mathsf{node}(\mathit{Left}, N, \mathit{Right}))}{\mathsf{below}(N, \mathit{Right})}$$

$$\frac{\mathsf{below}(N, \mathsf{node}(\mathit{Left}, K, \mathit{Right}))}{\mathsf{leq}(N, K)} \qquad \frac{\mathsf{below}(N, \mathsf{node}(\mathit{Left}, K, \mathit{Right}))}{\mathsf{ord}(\mathsf{node}(\mathit{Left}, K, \mathit{Right}))}$$

## 4 Deriving Inference Rules (30 pts)

We can use forward chaining in order to derive "big-step" inference rules from propositions. Recall a few characteristic rules for forward chaining, where all atoms are positive.

$$\frac{\Gamma, [D(X)] \xrightarrow{f} G^+}{\Gamma, [\forall x.D(x)^-] \xrightarrow{f} G^+}\ \forall L^* \qquad \frac{\Gamma, P^+ \xrightarrow{f} G^+}{\Gamma, [\uparrow P^+] \xrightarrow{f} G^+}\ \uparrow L \qquad \frac{}{\Gamma, P^+ \xrightarrow{f} [P^+]}\ \mathsf{id}^+$$

$$\frac{\Gamma, [D_1^-] \xrightarrow{f} G^+}{\Gamma, [D_1^- \wedge D_2^-] \xrightarrow{f} G^+}\ \wedge L_1 \qquad \frac{\Gamma, [D_2^-] \xrightarrow{f} G^+}{\Gamma, [D_1^- \wedge D_2^-] \xrightarrow{f} G^+}\ \wedge L_2 \qquad \frac{\Gamma \xrightarrow{f} [G_1^+] \quad \Gamma, [D_2^-] \xrightarrow{f} G^+}{\Gamma, [G_1^+ \supset D_2^-] \xrightarrow{f} G^+}\ \supset L$$

where $X$ is a freshly chosen variable subject to unification in $\forall L$.

**Task 1** (5 pts). When is an inference rule with judgment $K$ in the premise and judgment $J$ in the conclusion *derivable*? Circle all that apply. **Correct: (i), (iii)**

(i) There is a hypothetical proof of the conclusion $J$ from hypothesis $K$.

(ii) If we add this rule to the system we do not change the provable judgments.

(iii) If we substitute a proof for $K$ in the premise we obtain a proof for $J$.

(iv) There is a differentiable function from $K$ to $J$.

(v) If we add the rule the system remains in harmony.

Now consider the negative propositions

$$\uparrow \mathsf{numbits}(\mathsf{e}, \mathsf{z}),$$
$$\forall n.\, \forall k.\, \mathsf{numbits}(n, k) \supset (\uparrow \mathsf{numbits}(\mathsf{b0}(n), \mathsf{s}(k)) \wedge \uparrow \mathsf{numbits}(\mathsf{b1}(n), \mathsf{s}(k)))$$

where all atomic propositions $\mathsf{numbits}(\_, \_)$ are *positive*. Using these two propositions as antecedents we can deduce $\mathsf{numbits}(n, k)$ iff the number of bits $\mathsf{b0}$ and $\mathsf{b1}$ in the *binary* number $n$ is the *unary* number $k$.

**Task 2** (5 pts). Derive all inference rules that arise from left focusing on $\uparrow \mathsf{numbits}(\mathsf{e}, \mathsf{z})$.

$$\frac{\Gamma, \mathsf{numbits}(\mathsf{e}, \mathsf{z}) \longrightarrow G^+}{\Gamma, [\uparrow \mathsf{numbits}(\mathsf{e}, \mathsf{z})] \longrightarrow G^+}\ \uparrow L \qquad \frac{\Gamma, \mathsf{numbits}(\mathsf{e}, \mathsf{z}) \longrightarrow G^+}{\Gamma \longrightarrow G^+}\ R_0$$

**Task 3** (15 pts). Derive all inference rules that arise from left focusing on

$$\forall n.\, \forall k.\, \mathsf{numbits}(n, k) \supset (\uparrow \mathsf{numbits}(\mathsf{b0}(n), \mathsf{s}(k)) \wedge \uparrow \mathsf{numbits}(\mathsf{b1}(n), \mathsf{s}(k)))$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{numbits}(N, K) \in \Gamma}{\Gamma \xrightarrow{f} [\mathsf{numbits}(N, K)]}\ \mathsf{id}^+ \quad \dfrac{\dfrac{\Gamma, \mathsf{numbits}(\mathsf{b0}(N), \mathsf{s}(K)) \xrightarrow{f} G^+}{\Gamma, [\uparrow \mathsf{numbits}(\mathsf{b0}(N), \mathsf{s}(K))] \xrightarrow{f} G^+}\uparrow L}{\Gamma, [(\uparrow \mathsf{numbits}(\mathsf{b0}(N), \mathsf{s}(K)) \wedge \uparrow \mathsf{numbits}(\mathsf{b1}(N), \mathsf{s}(K)))] \xrightarrow{f} G^+}\wedge L_1}{\Gamma, [\mathsf{numbits}(N, K) \supset (\uparrow \mathsf{numbits}(\mathsf{b0}(N), \mathsf{s}(K)) \wedge \uparrow \mathsf{numbits}(\mathsf{b1}(N), \mathsf{s}(K)))] \xrightarrow{f} G^+}\supset L}{\Gamma, [\forall n.\, \forall k.\, \mathsf{numbits}(n, k) \supset (\uparrow \mathsf{numbits}(\mathsf{b0}(n), \mathsf{s}(k)) \wedge \uparrow \mathsf{numbits}(\mathsf{b1}(n), \mathsf{s}(k)))] \xrightarrow{f} G^+}\forall L \times 2$$

$$\dfrac{\Gamma, \mathsf{numbits}(N, K), \mathsf{numbits}(\mathsf{b0}(N), \mathsf{s}(K)) \longrightarrow G^+}{\Gamma, \mathsf{numbits}(N, K) \longrightarrow G^+}\ R_1$$

and, symmetrically, using $\wedge L_2$ instead of $\wedge L_1$:

$$\dfrac{\Gamma, \mathsf{numbits}(N, K), \mathsf{numbits}(\mathsf{b1}(N), \mathsf{s}(K)) \longrightarrow G^+}{\Gamma, \mathsf{numbits}(N, K) \longrightarrow G^+}\ R_2$$

**Task 4** (5 pts). Use your rules to prove that $\mathsf{numbits}(\mathsf{b0}(\mathsf{b1}(\mathsf{e})), \mathsf{s}(\mathsf{s}(\mathsf{z})))$, that is, the number of bits in the binary number 2 is 2.

Define $G_0 = \mathsf{numbits}(\mathsf{b0}(\mathsf{b1}(\mathsf{e})), \mathsf{s}(\mathsf{s}(\mathsf{z})))$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\mathsf{numbits}(\mathsf{e}, \mathsf{z}), \mathsf{numbits}(\mathsf{b1}(\mathsf{e}), \mathsf{s}(\mathsf{z})), \mathsf{numbits}(\mathsf{b0}(\mathsf{b1}(\mathsf{e})), \mathsf{s}(\mathsf{s}(\mathsf{z}))) \longrightarrow G_0}\ \mathsf{id}}{\mathsf{numbits}(\mathsf{e}, \mathsf{z}), \mathsf{numbits}(\mathsf{b1}(\mathsf{e}), \mathsf{s}(\mathsf{z})) \longrightarrow G_0}\ R_1}{\mathsf{numbits}(\mathsf{e}, \mathsf{z}) \longrightarrow G_0}\ R_2}{\cdot \longrightarrow G_0}\ R_0$$

To just use derived rules, we should also derive a rule by focusing on $G_0$ on the right. This new rule

$$\dfrac{}{\Gamma, \mathsf{numbits}(\mathsf{b0}(\mathsf{b1}(\mathsf{e})), \mathsf{s}(\mathsf{s}(\mathsf{z}))) \longrightarrow G_0}\ R_3$$

would just replace the use of id in the last step.