Final Exam

15-317: Constructive Logic
Frank Pfenning, Ron Garcia, William Lovas

December 14, 2009

Name: Andrew ID:

Instructions

- This exam is open-book.
- You have 3 hours to complete the exam.
- There are 6 problems.
- Remember to label all inference rules in your deductions.
- Consider writing out deductions on scratch paper first.

	Temporal Logic	Sequent Calculus	Prolog	Bottom-Up LP	Linear LP	Logical Rep'n	
	Prob 1	Prob 2	Prob 3	Prob 4	Prob 5	Prob 6	Total
Score							
Max	50	60	40	50	50	50	300
Grader							

1 Temporal Logic (50 pts)

In intuitionistic logic, we were mainly concerned with the judgment A *true*; in classical logic we added A *false*. In this problem we explore some aspects of *temporal logic*, which is based on the judgment $A \mathbin{@} t$, which we read as A *is true at time* t.

The rules for all the connectives and quantifiers are all relativized to arbitrary points t in time. For example:

$$\frac{\begin{array}{c}\overline{A \mathbin{@} t}\ u \\ \vdots \\ B \mathbin{@} t\end{array}}{A \supset B \mathbin{@} t}\ \supset I^u \qquad \frac{A \supset B \mathbin{@} t \quad A \mathbin{@} t}{B \mathbin{@} t}\ \supset E^u$$

We also have two new forms of propositions:

$\Box A$	A will be true at *all future* times
$\Diamond A$	A will be true at *some future* time

$\Box$ and $\Diamond$ are called *modalities* or *modal operators*. They are similar to universal and existential quantification, but apply to time. This similarity manifests itself in the introduction and elimination rules.

The flow of time is represented by a new judgment, $t \leq t'$ which means that t *comes before* t'. The flow of time is reflexive and transitive; otherwise we hold time abstract.

$$\frac{}{t \leq t}\ \mathsf{refl} \qquad \frac{t \leq t' \quad t' \leq t''}{t \leq t''}\ \mathsf{trans}$$

With this preparation, we can now give the introduction and elimination rules for $\Box A$. We write c for a new time parameter.

$$\frac{\begin{array}{c}\overline{t \leq c} \\ \vdots \\ A \mathbin{@} c\end{array}}{\Box A \mathbin{@} t}\ \Box I^c \qquad \frac{\Box A \mathbin{@} t \quad t \leq t'}{A \mathbin{@} t'}\ \Box E$$

In the $\Box I$ rule, c denotes a new time parameter, and the assumption $t \leq c$ indicates that c is later than t. This assumption is available only in the proof of $A \mathbin{@} c$. In other words, the judgment in the premise of $\Box I$ is parametric in c and hypothetical in the assumption $t \leq c$. This gives rise to substitution principles you may use freely as needed.

Task 1 (10 pts). Show that the elimination rule for $\Box$ is locally sound with respect to the introduction rule.

$$\dfrac{\dfrac{\dfrac{\overline{t \leq c}}{\mathcal{D}}}{\dfrac{A \mathbin{@} c}{\Box A \mathbin{@} t}\,\Box I^c} \quad \dfrac{\mathcal{E}}{t \leq t'}}{A \mathbin{@} t'}\,\Box E \qquad \Longrightarrow_R \qquad \dfrac{\dfrac{\mathcal{E}}{t \leq t'}}{\dfrac{[t'/c]\mathcal{D}}{A \mathbin{@} t'}}$$

Task 2 (10 pts). Show that the elimination rule for $\Box$ is locally complete with respect to the introduction rule.

$$\dfrac{\mathcal{D}}{\Box A \mathbin{@} t} \qquad \Longrightarrow_E \qquad \dfrac{\dfrac{\mathcal{D}}{\Box A \mathbin{@} t} \quad \dfrac{\overline{t \leq c}}{}}{\dfrac{A \mathbin{@} c}{\Box A \mathbin{@} t}\,\Box I^c}\,\Box E$$

Task 3 (10 pts). Prove that $(\Box A) \supset (\Box\Box A)$ @ t_0, that is, if A will be true at all future times then "A will be true at all future times" will be true at all future times.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Box A \ @ \ t_0}\, u \quad \dfrac{\dfrac{}{t_0 \leq c} \quad \dfrac{}{c \leq d}}{t_0 \leq d}\, \text{trans}}{A \ @ \ d}\, \Box E}{\Box A \ @ \ c}\, \Box I^d}{\Box\Box A \ @ \ t_0}\, \Box I^c}{\Box A \supset \Box\Box A \ @ \ t_0}\, \supset I^u$$

Task 4 (10 pts). Prove that $\Box(A \supset B) \supset ((\Box A) \supset \Box B)$ @ t_0, that is, if A implies B at all future times, then if A will be true at all future times then B will be true at all future times.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Box(A \supset B) \ @ \ t_0}\, u \quad \dfrac{}{t_0 \leq c}}{A \supset B \ @ \ c}\, \Box E \quad \dfrac{\dfrac{}{\Box A \ @ \ t_0}\, w \quad \dfrac{}{t_0 \leq c}}{A \ @ \ c}\, \Box E}{B \ @ \ c}\, \supset E}{\Box B \ @ \ t_0}\, \Box I^c}{\Box A \supset \Box B \ @ \ t_0}\, \supset I^w}{\Box(A \supset B) \supset \Box A \supset \Box B \ @ \ t_0}\, \supset I^u$$

Task 5 (10 pts). $\Diamond A$ is true if A will be true at *some* future time. Here is the introduction rule:

$$\frac{t \leq t' \quad A \mathbin{@} t'}{\Diamond A \mathbin{@} t}\ \Diamond I$$

Give the elimination rule for $\Diamond A$. **Hint:** The $\Diamond$ operator quantifies existentially over a future time, so we would expect the $\Diamond E$ rule to be somewhat analogous to the $\exists E$ rule.

$$\frac{\Diamond A \mathbin{@} t \qquad \begin{array}{c} \overline{t \leq c} \quad \overline{A \mathbin{@} c}^{\,u} \\ \vdots \\ C \mathbin{@} s \end{array}}{C \mathbin{@} s}\ \Diamond E^{c,u}$$

2 Sequent Calculus (60 pts)

In a sequent calculus, we make all assumptions explicit in the main judgment. The general form of a sequent for temporal logic now has the form

$$\underbrace{t_1 \le t_1', \ldots, t_k \le t_k'}_{\Theta} \; ; \; \underbrace{A_1 \mathbin{@} s_1, \ldots, A_n \mathbin{@} s_n}_{\Gamma} \Longrightarrow C \mathbin{@} s$$

where all t's and s's are times. In addition we have a hypothetical judgment $\Theta \models t \le t'$ which means that the assumptions in Θ entail that t comes before t'. We can define this simply by saturation: $\Theta \models t \le t'$ if either $t = t'$ or $t \le t'$ is in the database created by saturating Θ under transitivity.

The left and right rules for the ordinary connectives are just relativized as before. As examples, we give the init rule, as well as right and left rules for implication.

$$\frac{(P \mathbin{@} t) \in \Gamma}{\Theta; \Gamma \Longrightarrow P \mathbin{@} t}\ \mathsf{init}$$

$$\frac{\Theta; (\Gamma, A \mathbin{@} t) \Longrightarrow B \mathbin{@} t}{\Theta; \Gamma \Longrightarrow A \supset B \mathbin{@} t}\ \supset R \qquad \frac{\Theta; (\Gamma, A \supset B \mathbin{@} t) \Longrightarrow A \mathbin{@} t \quad \Theta; (\Gamma, A \supset B \mathbin{@} t, B \mathbin{@} t) \Longrightarrow C \mathbin{@} s}{\Theta; (\Gamma, A \supset B \mathbin{@} t) \Longrightarrow C \mathbin{@} s}\ \supset L$$

Here are the right and left rules for $\Diamond$.

$$\frac{\Theta \models t \le t' \quad \Theta; \Gamma \Longrightarrow A \mathbin{@} t'}{\Theta; \Gamma \Longrightarrow \Diamond A \mathbin{@} t}\ \Diamond R \qquad \frac{(\Theta, t \le c); (\Gamma, \Diamond A \mathbin{@} t, A \mathbin{@} c) \Longrightarrow C \mathbin{@} s}{\Theta; (\Gamma, \Diamond A \mathbin{@} t) \Longrightarrow C \mathbin{@} s}\ \Diamond L^c$$

Task 1 (10 pts). Give a sequent derivation of $(\cdot); (\cdot) \Longrightarrow P \supset \Diamond P \mathbin{@} t_0$.

$$\dfrac{\dfrac{\dfrac{\overline{\cdot \models t_0 \le t_0} \quad \overline{\cdot; P \mathbin{@} t_0 \Longrightarrow P \mathbin{@} t_0}\ \mathsf{init}}{\cdot; P \mathbin{@} t_0 \Longrightarrow \Diamond P \mathbin{@} t_0}\ \Diamond R}{\cdot; \cdot \Longrightarrow P \supset \Diamond P \mathbin{@} t_0}\ \supset R$$

Task 2 (10 pts). Prove that $(\cdot);(\cdot) \Longrightarrow \Diamond A \supset A \mathbin{@} t_0$ does not hold in general.

Applying inference rules bottom-up, we are forced into a deduction of the following form:

$$\dfrac{\dfrac{\dfrac{\vdots}{(t_0 \leq c, t_0 \leq d); (\Diamond A \mathbin{@} t_0, A \mathbin{@} c, A \mathbin{@} d) \Longrightarrow A \mathbin{@} t_0}\, \Diamond L^d}{t_0 \leq c; (\Diamond A \mathbin{@} t_0, A \mathbin{@} c) \Longrightarrow A \mathbin{@} t_0}\, \Diamond L^c}{\cdot; \Diamond A \mathbin{@} t_0 \Longrightarrow A \mathbin{@} t_0}\, \supset R}{\cdot; \cdot \Longrightarrow \Diamond A \supset A \mathbin{@} t_0}$$

At no point will we be able to use any of the assumptions $A \mathbin{@} c, A \mathbin{@} d, \ldots$ to prove $A \mathbin{@} t_0$, for example, when A is an atom P.

Task 3 (20 pts). Give the $\Box R$ and $\Box L$ rules.

$$\dfrac{\Theta, t \leq c; \Gamma \Longrightarrow A \mathbin{@} c}{\Theta; \Gamma \Longrightarrow \Box A \mathbin{@} t}\, \Box R^c \qquad \dfrac{\Theta \models t \leq t' \quad \Theta; (\Gamma, \Box A \mathbin{@} t, A \mathbin{@} t') \Longrightarrow C \mathbin{@} s}{\Theta; (\Gamma, \Box A \mathbin{@} t) \Longrightarrow C \mathbin{@} s}\, \Box L$$

Task 4 (10 pts). State the cut principle for temporal sequent calculus.

If $\Theta; \Gamma \Longrightarrow A \mathbin{@} t$ and $\Theta; \Gamma, A \mathbin{@} t \Longrightarrow C \mathbin{@} s$ then $\Theta; \Gamma \Longrightarrow C \mathbin{@} s$.

Task 5 (10 pts). The identity principle for temporal sequent calculus states that $(\cdot); A \mathbin{@} t \Longrightarrow A \mathbin{@} t$ for all A and t. Show the case in its proof concerning $A = \Diamond A'$. You may freely use weakening principles without proof.

$\cdot; A' \mathbin{@} c \Longrightarrow A' \mathbin{@} c$	By i.h. on A'
$t \leq c; \Diamond A' \mathbin{@} t, A' \mathbin{@} c \Longrightarrow A' \mathbin{@} c$	By weakening
$t \leq c \models t \leq c$	By definition of $\models$
$t \leq c; \Diamond A' \mathbin{@} t, A' \mathbin{@} c \Longrightarrow \Diamond A' \mathbin{@} t$	By $\Diamond R$
$\cdot; \Diamond A' \mathbin{@} t \Longrightarrow \Diamond A' \mathbin{@} t$	By $\Diamond L$

3 Prolog (40 pts)

In this problem we explore binary trees representing sets of integers (without duplication), represented as a type $tree$. We have two kinds of terms:

leaf	A leaf in the binary tree
node(l, x, r)	A node with left subtree l, integer x, and right subtree r.

The important data structure invariant is that for any node node(l, x, r) all integers in the left subtree l are strictly smaller than x, and all integers in the right subtree r are strictly greater than x. We are not concerned with whether the tree is balanced.

Task 1 (10 pts). Write a Prolog program smallest$(+tree, -int)$ that returns the smallest element in the tree. Your program should exploit the representation invariant.

smallest(node(leaf, X, R), X).
smallest(node(L, X, R), Y) $\leftarrow$ smallest(L, Y).

Task 2 (20 pts). Write a Prolog program insert$(+int, +tree, -tree)$, where insert(y, t, u) inserts an integer y into the tree t and returns the result as u, where u must still satisfy the ordering invariant. You may use the built-in comparisons $x < y$, $x = y$, $x > y$ on ground integers, but your program should be pure (that is, not use cut).

insert(Y, leaf, node(leaf, Y, leaf)).

insert(Y, node(L, X, R), node(L_1, X, R)) $\leftarrow$
    $Y < X$, insert(Y, L, L_1).

insert(Y, node(L, X, R), node(L, X, R)) $\leftarrow$
    $Y = X$.

insert(Y, node(L, X, R), node(L, X, R_1)) $\leftarrow$
    $Y > X$, insert(Y, R, R_1).

Task 3 (5 pts). Can your program be run in the mode $\mathsf{insert}(+int, -tree, +tree)$ to remove an element from the tree? Explain briefly why or why not.

The program is well-moded, but it will not remove the element unless it is stored in a node just above two leaves.

Task 4 (5 pts). Add cuts to your program for insert to improve its efficiency under the mode $\mathsf{insert}(+int, +tree, -tree)$. The cuts must be *green cuts* in this mode.

$\mathsf{insert}(Y, \mathsf{leaf}, \mathsf{node}(\mathsf{leaf}, Y, \mathsf{leaf})).$

$\mathsf{insert}(Y, \mathsf{node}(L, X, R), \mathsf{node}(L_1, X, R)) \leftarrow$
$\quad Y < X, !, \mathsf{insert}(Y, L, L_1).$

$\mathsf{insert}(Y, \mathsf{node}(L, X, R), \mathsf{node}(L, X, R)) \leftarrow$
$\quad Y = X, !.$

$\mathsf{insert}(Y, \mathsf{node}(L, X, R), \mathsf{node}(L, X, R_1)) \leftarrow$
$\quad Y > X, !, \mathsf{insert}(Y, R, R_1).$

4 Bottom-Up Logic Programming (50 pts)

We return to the representation of binary trees representing sets of integers from Problem 3. In this problem we assume all predicates are persistent. We suggest you formulate your bottom-up logic programs as inference rules, but you may also use logical propositions under the forward-chaining interpretation. You should exploit the ordering invariant on the trees where possible.

Task 1 (10 pts). Assume we have a predicate $\mathsf{root}(t)$ which is asserted for a binary tree t representing a set of integers. Write a saturating, bottom-up logic program to deduce $\mathsf{subtree}(s)$ for every subtree s of t (including t itself).

$$\frac{\mathsf{root}(t)}{\mathsf{subtree}(t)} \qquad \frac{\mathsf{subtree}(l, x, r)}{\begin{array}{l}\mathsf{subtree}(l)\\ \mathsf{subtree}(r)\end{array}}$$

Task 2 (15 pts). For each subtree, as generated by your code from Task 1, compute a predicate $\mathsf{size}(t, k)$ which means that there are k elements in the subtree t. You may use premises of the form $x = e$ to compute the arithmetic expression e (which must be ground) and then match it against x (which may be ground or a variable). Your code may use subtree.

$$\frac{}{\mathsf{size}(\mathsf{leaf}, 0)} \qquad \frac{\begin{array}{l}\mathsf{subtree}(\mathsf{node}(l, x, r))\\ \mathsf{size}(l, kl)\\ \mathsf{size}(r, kr)\\ k = kl + kr + 1\end{array}}{\mathsf{size}(\mathsf{node}(l, x, r), k)}$$

Task 3 (15 pts). For each element x in the tree, compute $\mathsf{smaller}(x, k)$ and $\mathsf{greater}(x, k)$, where k is the *number of elements* strictly smaller and greater than x in the tree, respectively. Your code may use subtree and size.

$$\frac{\mathsf{root}(t)}{\mathsf{greater}(t, 0)\quad \mathsf{smaller}(t, 0)} \qquad \frac{\mathsf{greater}(\mathsf{node}(l, x, r), k_1)\quad \mathsf{size}(r, k_2)\quad k = k_1 + k_2\quad k' = k + 1}{\mathsf{greater}(x, k)\quad \mathsf{greater}(l, k')} \qquad \frac{\mathsf{smaller}(\mathsf{node}(l, x, r), k_1)\quad \mathsf{size}(l, k_2)\quad k = k_1 + k_2\quad k' = k + 1}{\mathsf{smaller}(x, k)\quad \mathsf{smaller}(r, k')}$$

Task 4 (10 pts). Compute $\mathsf{median}(x)$, where x is the median of the elements in the tree. Assume that you have an odd number of elements. Your code may use subtree, size, smaller, and greater.

$$\frac{\mathsf{smaller}(x, k)\quad \mathsf{greater}(x, k)}{\mathsf{median}(x)}$$

5 Linear Logic Programming (50 pts)

The 16-puzzle consists of a 4×4 square filled with 15 tiles, numbered 1 through 15, and one empty space. Tiles can slide horizontally or vertically into the empty space. For example, in the position on the left there are three possible moves, one of which transforms

1	2	3	4
5	6	7	8
9	10	11	12
13	14		15

into

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

The goal is to achieve the situation depicted on the right, from some initial configuration.

We represent this with the following emphemeral predicates.

$\mathsf{sq}(x, y, n)$	Square (x, y) has tile n.
$\mathsf{empty}(x, y)$	Square (x, y) is empty.

The upper left-hand corner is $(0, 0)$, the lower right-hand corner is $(\mathsf{s}(\mathsf{s}(\mathsf{s}(0))), \mathsf{s}(\mathsf{s}(\mathsf{s}(0))))$, with x increasing left-to-right and y increasing top-to-bottom. All numbers are represented in unary form, using 0 and $\mathsf{s}(-)$.

Task 1 (20 pts). Represent the legal moves. You may use a representation by logical formulas (in linear logic) or by inference rule with ephemeral premises and conclusions. Your rules should not depend on the size of the board.

right	:	$\mathsf{sq}(x, y, n) \otimes \mathsf{empty}(\mathsf{s}(x), y) \multimap \mathsf{empty}(x, y) \otimes \mathsf{sq}(\mathsf{s}(x), y, n)$
left	:	$\mathsf{empty}(x, y) \otimes \mathsf{sq}(\mathsf{s}(x), y, n) \multimap \mathsf{sq}(x, y, n) \otimes \mathsf{empty}(\mathsf{s}(x), y)$
down	:	$\mathsf{sq}(x, y, n) \otimes \mathsf{empty}(x, \mathsf{s}(y)) \multimap \mathsf{empty}(x, y) \otimes \mathsf{sq}(x, \mathsf{s}(y), n)$
up	:	$\mathsf{empty}(x, y) \otimes \mathsf{sq}(x, \mathsf{s}(y), n) \multimap \mathsf{sq}(x, y, n) \otimes \mathsf{empty}(x, \mathsf{s}(y))$

Task 2 (5 pts). We call a state Δ *valid* if it represents a proper board configuration with tiles 1–15 and one empty square, regardless whether the puzzle has a solution from that configuration. If viewed as a bottom-up linear logic program, will your representation reach *quiescence* when starting from a valid state? Circle one of the following.

(i) Always (ii) Sometimes (iii) Never

Never

Task 3 (20 pts). Write a program to recognize the winning position for an arbitrary board with $c > 1$ columns and $r > 1$ rows. You should assume persistent facts

$!\mathsf{maxcol}(c-1)$
$!\mathsf{maxrow}(r-1)$
$!\mathsf{maxtile}(c \times r - 1)$

where $c-1$, $r-1$ and $c \times r - 1$ are precomputed in unary notation using 0 and $\mathsf{s}(-)$. You may use persistent predicates $x = y$ and $x \neq y$ for ground x and y (as in Lollibot).

Your program should have the property that a state $\Delta \longrightarrow^* \mathsf{winning}$ if and only if Δ represents the winning configuration. If it is not winning, it may get stuck in an arbitrary state.

$\mathsf{empty}(\mathsf{s}(x), y) \otimes !\mathsf{maxtile}(n) \multimap \mathsf{consume}(x, y, n).$

$\mathsf{consume}(\mathsf{s}(x), y, \mathsf{s}(n)) \otimes \mathsf{sq}(\mathsf{s}(x), y, \mathsf{s}(n)) \multimap \mathsf{consume}(x, y, n).$

$\mathsf{consume}(0, \mathsf{s}(y), \mathsf{s}(n)) \otimes \mathsf{sq}(0, \mathsf{s}(y), \mathsf{s}(n)) \otimes !\mathsf{maxcol}(c) \multimap \mathsf{consume}(c, y, n).$

$\mathsf{consume}(0, 0, \mathsf{s}(0)) \otimes \mathsf{sq}(0, 0, \mathsf{s}(0)) \multimap \mathsf{winning}.$

Task 4 (5 pts). If viewed as a bottom-up linear logic program, will your program reach quiescence when starting from a valid state? Circle one of the following.

(i) Always (ii) Sometimes (iii) Never

Always

6 Logical Representation (50 pts)

Write each of the following sentences in logical form, given the listed vocabulary. Not all of the vocabulary may be necessary to render each example. We have filled in one example for you.

> And my head I'd be scratchin'
> While my thoughts were busy hatchin'
> if I only had a brain.
> — Scarecrow in *The Wizard of Oz*

- **Types:** $person$, $object$.
- **Terms:** $\mathsf{scarecrow} : person$, $\mathsf{brain} : object$, $\mathsf{head} : object$.
- **Predicates:** $\mathsf{has}(person, object)$, $\mathsf{thinking}(person)$, $\mathsf{scratching}(person, object)$
- **Connectives:** $\Box A$, $\Diamond A$, $A \supset B$, $A \wedge B$.

$\Box(\mathsf{has}(\mathsf{scarecrow}, \mathsf{brain}) \supset \Box(\mathsf{thinking}(\mathsf{scarecrow}) \supset \mathsf{scratching}(\mathsf{scarecrow}, \mathsf{head})))$
or
$\Box(\mathsf{has}(\mathsf{scarecrow}, \mathsf{brain}) \supset \Diamond(\mathsf{thinking}(\mathsf{scarecrow}) \wedge \Box(\mathsf{thinking}(\mathsf{scarecrow}) \supset \mathsf{scratching}(\mathsf{scarecrow}, \mathsf{head}))))$

Due to ambiguity in natural language, one should not expect the answers to be unique. Try to make your logical expression as close as you can.

Task 1 (10 pts).

> The sad truth is that opportunity doesn't knock twice.
> — Gloria Estefan

- **Types:** $agent$.
- **Terms:** $\mathsf{opportunity} : agent$.
- **Predicates:** $\mathsf{knock}(agent)$.
- **Connectives:** $\Box A$, $\Diamond A$, $A \supset B$, $\neg A$.
- **Note:** For simplicity, assume for this task that flow of time is irreflexive, that is, the future does not include the present.

$\Box(\mathsf{knock}(\mathsf{opportunity}) \supset \neg\Diamond\mathsf{knock}(\mathsf{opportunity}))$
or
$\Box(\mathsf{knock}(\mathsf{opportunity}) \supset \Box\neg\mathsf{knock}(\mathsf{opportunity}))$

Task 2 (10 pts).

> If wishes were horses, beggars would ride.
> — English proverb

- **Types:** $\mathit{anything}$.
- **Terms:** none.
- **Predicates:** $\mathsf{wish}(\mathit{anything})$, $\mathsf{horse}(\mathit{anything})$, $\mathsf{beggar}(\mathit{anything})$, $\mathsf{ride}(\mathit{anything})$.
- **Connectives:** $\forall x{:}\tau.A$, $\exists x{:}\tau.A$, $A \supset B$, $A \wedge B$, $\top$.

$$(\forall x{:}\mathit{anything}.\ \mathsf{wish}(x) \supset \mathsf{horse}(x)) \supset (\forall x{:}\mathit{anything}.\ \mathsf{beggar}(x) \supset \mathsf{ride}(x))$$

Task 3 (10 pts).

> If I had a nickel for every time I had a nickel, I would have two nickels.
> — Anonymous (who did not take 15-317)

- **Types:** person, object
- **Terms:** $\mathsf{anon} : \mathit{person}$, $\mathsf{nickel} : \mathit{object}$.
- **Predicates:** $\mathsf{has}(\mathit{person}, \mathit{object})$.
- **Connectives:** $!A$, $A \multimap B$, $A \otimes B$.
- **Hint:** In linear logic $A \supset B$ can be defined as $(!A) \multimap B$.

$$(!(\mathsf{has}(\mathsf{anon}, \mathsf{nickel}) \multimap \mathsf{has}(\mathsf{anon}, \mathsf{nickel}))) \multimap \mathsf{has}(\mathsf{anon}, \mathsf{nickel}) \otimes \mathsf{has}(\mathsf{anon}, \mathsf{nickel})$$

Task 4 (10 pts).

> If I had a nickel for every time a student asked me for an extension on a homework, I would be a millionaire.
> — Frank Pfenning

- **Types:** $favor$, $person$, $object$.
- **Terms:** $\mathsf{extension} : favor$, $\mathsf{fp} : person$, $\$0.05 : object$, $\$1M : object$
- **Predicates:** $\mathsf{student}(person)$, $\mathsf{ask}(person, person, favor)$, $\mathsf{has}(person, object)$.
- **Connectives:** $!A$, $A \multimap B$, $A \otimes B$, $\exists x{:}\tau.A$, $\forall x{:}\tau.A$.

$$(!((\exists s{:}person.\, !\mathsf{student}(s) \otimes \mathsf{ask}(s, \mathsf{fp}, \mathsf{extension})) \multimap \mathsf{has}(\mathsf{fp}, \$0.05))) \multimap \mathsf{has}(\mathsf{fp}, \$1M)$$

Task 5 (10 pts).

> If you can't stand the heat, get out of the kitchen.
> — Harry S. Truman

- **Types:** $condition$, $location$, $person$.
- **Terms:** $\mathsf{heat} : condition$, $\mathsf{kitchen} : location$.
- **Predicates:** $\mathsf{withstand}(person, condition)$, $\mathsf{in}(person, location)$.
- **Connectives:** $!A$, $A \multimap B$, $\mathbf{1}$, $\mathbf{0}$, $\forall x{:}\tau.A$.
- **Hint:** In linear logic, $\neg A$ can be defined as $A \multimap \mathbf{0}$.

$$\forall x{:}person.(\mathsf{withstand}(x, \mathsf{heat}) \multimap \mathbf{0}) \multimap (\mathsf{in}(x, \mathsf{kitchen}) \multimap \mathbf{1})$$
or
$$\forall x{:}person.!(\mathsf{withstand}(x, \mathsf{heat}) \multimap \mathbf{0}) \multimap (\mathsf{in}(x, \mathsf{kitchen}) \multimap \mathbf{1})$$