Assignment 1: Induction

15-312: Foundations of Programming Languages
Kevin Watkins (`kw@cmu.edu`)

Out: Thursday, 20 Jan 2005
Due: Thursday, 27 Jan 2005 **(10:30 am)**

50 points total

Welcome to 15-312! This assignment is intended to give you exercise with the kind of thinking that we'll do in this course. You are encouraged, but not required, to typeset your answers; if you write by hand, write legibly. The assignment is due at the *beginning of lecture* on Thursday, January 27.

Credit in this assignment will reflect the clarity and mathematical rigor with which you present your arguments. A proof should be self-evident. Although some things may seem obvious, please take care to be precise and complete.

Be sure that what you say in your proof is correct. **Less credit** will be given for incorrect steps than for missing steps. You will receive more points turning in an explanation of your strategy and leaving blank parts where you couldn't finish the proof, than turning in a complete proof with incorrect steps.

Of course, if you have any questions, feel free to e-mail Kevin Watkins (`kw@cmu.edu`) or come to office hours!

All work must be your own. Please read the collaboration policy at

`http://www.cs.cmu.edu/~crary/312/assignments.html`

1 Higher-order abstract syntax (5 points)

In recitation we saw higher-order abstract syntax for the integral operator $\int_a^b e\,dx$ of calculus.

Question 1.1 (5 points).

Using our standard notations for higher-order abstract syntax with names:

$$op(e_1, \ldots, e_n), \quad x, \quad x.e,$$

fill in the rest of a grammar for a language of calculus expressions including operators for $e_1 + e_2$, $e_1 \cdot e_2$, $\int_a^b e\,dx$, and $\sum_{i=a}^b e$. The first part is done for you:

$$e \quad ::= \quad \text{plus}(e_1, e_2) \mid \ldots$$

Question 1.2 (EXTRA CREDIT).

Extend the grammar with higher-order abstract syntax for a derivative operator. (This is tricky!)

2 Substitution theorem (25 points)

An important theorem of a programming language is the *substitution theorem*. Informally, it states that the result of substitution of a well-typed term for a free variable of another well-typed term is itself well-typed. In order to be able to state the substitution theorem, we need a definition of well-typedness and a definition of substitution. In class we wrote substitution as a partial function. Here we want to prove a property of substitution by rule induction, so we will define it by a judgment with inference rules instead.

Here are the judgments that we define:

$\Gamma \vdash e : \texttt{int}$	In the context Γ, e has type `int` (the only type of our language).
$\texttt{FV}(e) = S$	S is the set of free variables in e.
$\{e_1/x\}e_2 = e_3$	Substituting e_1 for free occurrences of x in e_2 results in expression e_3.

Here is the grammar for our language:

$$\begin{array}{lcl}
\Gamma & ::= & \cdot \mid \Gamma, x : \texttt{int} \\
e & ::= & k \mid e_1 + e_2 \mid x \mid \texttt{let}\, x = e_1 \,\texttt{in}\, e_2 \,\texttt{end} \\
k & ::= & 0 \mid 1 \mid \ldots \mid 45 \mid \ldots \\
x & ::= & \text{(variables)}
\end{array}$$

Here is the definition of well-formedness:

$$\frac{}{\Gamma, x : \texttt{int}, \Gamma' \vdash x : \texttt{int}}\ \text{var} \qquad \frac{}{\Gamma \vdash k : \texttt{int}}\ \text{num}$$

$$\frac{\Gamma \vdash e_1 : \texttt{int} \quad \Gamma \vdash e_2 : \texttt{int}}{\Gamma \vdash e_1 + e_2 : \texttt{int}}\ \text{plus} \quad \frac{\Gamma \vdash e_1 : \texttt{int} \quad \Gamma, x : \texttt{int} \vdash e_2 : \texttt{int}}{\Gamma \vdash \texttt{let}\, x = e_1\, \texttt{in}\, e_2\, \texttt{end} : \texttt{int}}\ \text{let}$$

Here are the definitions of free variables and substitution:

$$\frac{}{\texttt{FV}(x) = \{x\}}\ \text{fv-var} \quad \frac{}{\texttt{FV}(k) = \{\}}\ \text{fv-num} \quad \frac{\texttt{FV}(e_1) = S_1 \quad \texttt{FV}(e_2) = S_2}{\texttt{FV}(e_1 + e_2) = S_1 \cup S_2}\ \text{fv-plus}$$

$$\frac{\texttt{FV}(e_1) = S_1 \quad \texttt{FV}(e_2) = S_2}{\texttt{FV}(\texttt{let}\, x = e_1\, \texttt{in}\, e_2\, \texttt{end}) = S_1 \cup (S_2 - \{x\})}\ \text{fv-let}$$

$$\frac{}{\{e/x\}x = e}\ \text{s-var=} \qquad \frac{x \neq y}{\{e/x\}y = y}\ \text{s-var} \neq$$

$$\frac{\{e/x\}e_1 = e_1' \quad \{e/x\}e_2 = e_2'}{\{e/x\}(e_1 + e_2) = (e_1' + e_2')}\ \text{s-plus} \qquad \frac{}{\{e/x\}k = k}\ \text{s-num}$$

$$\frac{\texttt{FV}(e) = S \quad y \notin S \cup \{x\} \quad \{e/x\}e_1 = e_1' \quad \{e/x\}e_2 = e_2'}{\{e/x\}(\texttt{let}\, y = e_1\, \texttt{in}\, e_2\, \texttt{end}) = (\texttt{let}\, y = e_1'\, \texttt{in}\, e_2'\, \texttt{end})}\ \text{s-let}$$

In order to prove the substitution theorem, we will need two lemmas. One, called *weakening*, roughly states that if an expression is well-typed in some context, we can add a variable to the context and still have a well-typed term, *as long as the new variable is not one of the free variables of e*. Weakening depends on the other lemma, called *exchange*, which tells us that we can swap the positions of two different variables anywhere in the context.

Here is the statement of the lemmas and a model proof of weakening. In the proof below, we write out each case and the steps we follow, alongside their justifications. (**wts** stands for "*want to show.*") You should follow this model when writing your own proofs.

Lemma 1 (Exchange)

If	$\Gamma, x : \texttt{int}, y : \texttt{int}, \Gamma' \vdash e : \texttt{int}$
and	$x \neq y$
then	$\Gamma, y : \texttt{int}, x : \texttt{int}, \Gamma' \vdash e : \texttt{int}$

Lemma 2 (Weakening)

If	$\Gamma \vdash e : \texttt{int}$
and	$\texttt{FV}(e) = S$ where $x \notin S$
then	$\Gamma, x : \texttt{int} \vdash e : \texttt{int}$.

Proof of weakening is by induction on the derivation of $\Gamma \vdash e$.

case	$\dfrac{}{\Gamma, y : \texttt{int}, \Gamma' \vdash y : \texttt{int}}\ \text{var}$	
wts	$\Gamma, y : \texttt{int}, \Gamma', x : \texttt{int} \vdash y : \texttt{int}$	
1	$\texttt{FV}(x) = \{x\}$	**(fv-var)**
2	$y \neq x$ because $y \notin \{x\}$	**(given)**
	$\Gamma, y : \texttt{int}, \Gamma', x : \texttt{int} \vdash y : \texttt{int}$	**(var)**[1]

case	$\dfrac{}{\Gamma \vdash k : \texttt{int}}\ \text{num}$	
wts	$\Gamma, x : \texttt{int} \vdash k : \texttt{int}$	
	$\Gamma, x : \texttt{int} \vdash k : \texttt{int}$	**(num)**

case	$\dfrac{\Gamma \vdash e_1 : \texttt{int} \quad \Gamma \vdash e_2 : \texttt{int}}{\Gamma \vdash e_1 + e_2 : \texttt{int}}\ \text{plus}$	
wts	$\Gamma, x : \texttt{int} \vdash e_1 + e_2 : \texttt{int}$	
1	$\texttt{FV}(e_1 + e_2) = S_1 \cup S_2$	**(inversion on fv-plus)**
	where $\texttt{FV}(e_1) = S_1$ and $\texttt{FV}(e_2) = S_2$	
2	$x \notin S_1 \cup S_2$	**(given)**
3	$x \notin S_1$	**(defn $\notin$, 2)**
4	$\Gamma, x : \texttt{int} \vdash e_1 : \texttt{int}$	**(induction on left premise, 3)**
5	$x \notin S_2$	**(defn $\notin$, 2)**
6	$\Gamma, x : \texttt{int} \vdash e_2 : \texttt{int}$	**(induction on right premise, 5)**
	$\Gamma, x : \texttt{int} \vdash e_1 + e_2 : \texttt{int}$	**(plus on 4,6)**

case	$\dfrac{\Gamma \vdash e_1 : \texttt{int} \quad \Gamma, y : \texttt{int} \vdash e_2 : \texttt{int}}{\Gamma \vdash \texttt{let}\, y = e_1\, \texttt{in}\, e_2\, \texttt{end} : \texttt{int}}\ \text{let}$	
wts	$\Gamma, x : \texttt{int} \vdash \texttt{let}\, y = e_1\, \texttt{in}\, e_2\, \texttt{end} : \texttt{int}$	
1	without loss of generality, say $x \neq y$	**(α-eq)**[2]
2	$\texttt{FV}(\texttt{let}\, y = e_1\, \texttt{in}\, e_2\, \texttt{end}) = S_1 \cup (S_2 - \{y\})$	**(inversion on fv-let)**
	where $\texttt{FV}(e_1) = S_1$ and $\texttt{FV}(e_2) = S_2$	
3	$x \notin S_1 \cup (S_2 - \{y\})$	**(given)**
4	$x \notin S_1$	**(defn $\notin$, 3)**
5	$x \notin S_2$	**(defn $\notin$, 3, 1)**
6	$\Gamma, x : \texttt{int} \vdash e_1 : \texttt{int}$	**(induction on left premise, 4)**
7	$\Gamma, y : \texttt{int}, x : \texttt{int} \vdash e_2 : \texttt{int}$	**(induction on right premise, 5)**
8	$\Gamma, x : \texttt{int}, y : \texttt{int} \vdash e_2 : \texttt{int}$	**(exchange on 7)**
	$\Gamma, x : \texttt{int} \vdash \texttt{let}\, y = e_1\, \texttt{in}\, e_2\, \texttt{end} : \texttt{int}$	**(let on 6,8)**

[1] We could have used this step directly. However, in a language with more than one type (as most every language we will see after this assignment) we want to make sure that the newly added variable doesn't shadow the existing variable at a different type. Therefore, we show here that the variables must be inequal, so that this proof looks more like it would in a typed language.

[2] Here's a nice trick. Since we consider all α-equivalent terms to be equal, we can simply choose y not equal to x and silently α-vary the term to agree with our choice.

This concludes the proof of weakening.

Theorem 1 (Substitution)

If	$\Gamma \vdash e_1 : \texttt{int}$
and	$x : \texttt{int}, \Gamma \vdash e_2 : \texttt{int}$
and	$\{e_1/x\}e_2 = e_3$
then	$\Gamma \vdash e_3 : \texttt{int}$.

Question 2.1 (25 points).

Prove the substitution theorem, using rule induction. You may use the weakening and exchange lemmas as stated, and do not have to prove them.

Hints: Think about which derivation to induct on. You should need weakening (but not exchange). If you don't, then you are not being careful enough or are on the wrong track. Remember the principle of *inversion*: if only one rule could have been used to derive a judgment that we know is derivable, then we know its premises must hold.

3 Propositional logic (20 points)

In this question we will look at a subset of *Propositional Logic*. Our universe of terms consists of an infinite number of arity 0 operators $P_0, P_1, \ldots, P_n$ ("propositional variables"), and the binary operator $\Rightarrow$ ("implication"). We write A, B, ... as metavariables standing for propositions. Thus, the grammar of the language is as follows.

$$A \quad ::= \quad P_i \mid A \Rightarrow A$$

Next we define a judgment A thm representing the assertion that A is a theorem of the logic.

$$\frac{}{A \Rightarrow (B \Rightarrow A)\ \mathsf{thm}}\ K$$

$$\frac{}{(A \Rightarrow (B \Rightarrow C)) \Rightarrow (A \Rightarrow B) \Rightarrow (A \Rightarrow C)\ \mathsf{thm}}\ S$$

$$\frac{A \Rightarrow B\ \mathsf{thm} \quad A\ \mathsf{thm}}{B\ \mathsf{thm}}\ App$$

Truth Value. If we have assignments (to `true` or `false`) for all of the propositional variables in a proposition, its truth value (either `true` or `false`) can be computed recursively using the following familiar truth table for $\Rightarrow$:

Proposition			**Truth Value**
`false`	$\Rightarrow$	`false`	`true`
`false`	$\Rightarrow$	`true`	`true`
`true`	$\Rightarrow$	`false`	`false`
`true`	$\Rightarrow$	`true`	`true`

Tautology. A proposition A is a tautology iff for every assignment giving a truth value `true` or `false` to each of the propositional variables $P_0, \dots, P_n$ appearing in A, the truth value of the whole proposition A is `true`.

Question 3.1 (5 points).

Rewrite the grammar for the language as a set of inference rules. The rules should inductively define the judgment A prop, which asserts that A is a well-formed proposition.

Question 3.2 (15 points).

Prove, using rule induction, that if A thm has a derivation then A is a tautology.

Question 3.3 (EXTRA CREDIT).

Find a proposition A that is a tautology, but not a theorem (that is, the judgment A thm cannot be derived). You do not need to prove that it is not a theorem!