Homework Assignment 8

Any automatically graded answer may be manually graded by the instructor. Submissions are expected to only use functions taught in the course. If a submission uses a disallowed function, that exercise can get zero points. Excluding promises, *all functions that mutate values are disallowed* (mutable functions usually have a `!` in their name).

Translating SimpleJS into LambdaJS

1. (90 points) Implement the following translation function from SimpleJS into LambdaJS.

$$
\begin{aligned}
\mathrm{J}[\![x.y]\!] &\stackrel{\text{def}}{=} (\texttt{deref}\ \mathsf{x})[\texttt{"y"}] \\
\mathrm{J}[\![x.y := e]\!] &\stackrel{\text{def}}{=} \texttt{let}\ \mathsf{data} = [\![e]\!]\ \texttt{in} \\
&\qquad \texttt{let}\ \mathsf{o} = (\texttt{deref}\ \mathsf{x})\ \texttt{in} \\
&\qquad\quad \mathsf{x} := \big(\mathsf{o}[\texttt{"y"}] \leftarrow \mathsf{data}\big); \\
&\qquad\quad \mathsf{data} \\
\mathrm{J}[\![x.y(e\cdots)]\!] &\stackrel{\text{def}}{=} \texttt{let}\ \mathsf{m} = (\texttt{deref}\ \mathsf{x})[\texttt{"y"}]\ \texttt{in} \\
&\qquad \texttt{let}\ \mathsf{f} = (\texttt{deref}\ \mathsf{m})[\texttt{"\$code"}]\ \texttt{in} \\
&\qquad\quad \mathsf{f}(\mathsf{x}, [\![e\cdots]\!]) \\
\mathrm{J}[\![\texttt{function}(x\cdots)\ \{e\}]\!] &\stackrel{\text{def}}{=} \texttt{alloc}\ \{\texttt{"\$code"} : \lambda(\mathsf{this}, \mathsf{x}\cdots).[\![e]\!], \texttt{"prototype"} : \texttt{alloc}\ \{\}\} \\
\mathrm{J}[\![\texttt{new}\ e_f(e\cdots)]\!] &\stackrel{\text{def}}{=} \texttt{let}\ \mathsf{ctor} = \texttt{deref}\ [\![e_f]\!]\ \texttt{in} \\
&\qquad \texttt{let}\ \mathsf{obj} = \texttt{alloc}\ \{\texttt{"\$proto"} : \mathsf{ctor}[\texttt{"prototype"}]\}\ \texttt{in} \\
&\qquad\quad \texttt{let}\ \mathsf{f} = \mathsf{ctor}[\texttt{"\$code"}]\ \texttt{in} \\
&\qquad\qquad \mathsf{f}(\mathsf{obj}, \mathrm{J}[\![e]\!]\cdots); \\
&\qquad\qquad \mathsf{obj} \\
\mathrm{J}[\![c]\!] &\stackrel{\text{def}}{=} c \\
\mathrm{J}[\![x]\!] &\stackrel{\text{def}}{=} \mathsf{x} \\
\mathrm{J}[\![\texttt{let}\ x = e_1\ \texttt{in}\ e_2]\!] &\stackrel{\text{def}}{=} \texttt{let}\ \mathsf{x} = [\![e_1]\!]\ \texttt{in}\ [\![e_2]\!]
\end{aligned}
$$

2. (5 points) Recall our discussion about the problem of capturing variables when generating code. Give **one** example of one incorrectly generated code if we use the rules above directly.

3. (5 points) **Manually graded.** Use `mk-let` instead of `j:let` to avoid the problem above.

Desugaring SimpleJS

4. (20 points) **Extra credit.** Implement the following desugaring function. *Hint #1:* `s:class-methods` is a map from `s:variable` into `s:function` and method `constructor` can be safely assumed to exist. *Hint #2:* Consider using function `s:begin` to assign each method to `cls`.

$$
\begin{aligned}
\mathrm{D}[\![\texttt{class extends } e\{\mathsf{ctor}(x\cdots)\{e_c\}\ y(z\cdots)\{e_m\}\cdots\}]\!] \stackrel{\text{def}}{=\!=} & \texttt{let } \mathsf{p_0} = \mathrm{D}[\![e]\!] \texttt{ in let } \mathsf{p_1} = \texttt{function}()\{\} \texttt{ in} \\
& \quad p_1.\mathsf{prototype} := p_0.\mathsf{prototype}; \\
& \quad \texttt{let } \mathsf{p_2} = \texttt{new } p_1 \texttt{ in let } \mathsf{cls} = \texttt{function}(x\cdots)\{e_c\} \texttt{ in} \\
& \quad\quad cls.\mathsf{prototype} := p_2; \\
& \quad\quad\quad p_2.m := \texttt{function}(y\cdots)\{e_m\}\cdots\ ; \\
& \quad\quad\quad cls \\
\mathrm{D}[\![\texttt{let } x = e_1 \texttt{ in } e_2]\!] \stackrel{\text{def}}{=\!=} & \texttt{let } x = \mathrm{D}[\![e_1]\!] \texttt{ in } \mathrm{D}[\![e_2]\!] \\
\mathrm{D}[\![x := e]\!] \stackrel{\text{def}}{=\!=} & x := \mathrm{D}[\![e]\!] \\
\mathrm{D}[\![\texttt{function}(x\cdots)\ \{e\}]\!] \stackrel{\text{def}}{=\!=} & x := \texttt{function}(x\cdots)\ \{\mathrm{D}[\![e]\!]\} \\
\mathrm{D}[\![\texttt{new } e_f(e\cdots)]\!] \stackrel{\text{def}}{=\!=} & \texttt{new } \mathrm{D}[\![e_f]\!](\mathrm{D}[\![e]\!]\cdots) \\
\mathrm{D}[\![x.y(e\cdots)]\!] \stackrel{\text{def}}{=\!=} & x.y(\mathrm{D}[\![e]\!]\cdots)
\end{aligned}
$$