Homework Assignment 9

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. Implement the following translation function from SimpleJS into LambdaJS. LambdaJS variables are underlined, while SimpleJS are not. We use the abstract-syntax notation for the let-binding, sequencing, the object constructor, and the function declaration (λ). We use indentation to highlight the scope of let-binders and of sequencing. You are encouraged to peruse `hw9-util.rkt`, as it gives usage examples and has helpful documentation to complete this assignment.

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