1 Macros for Liquid Equality Chains

An $\langle expr\rangle$ is a well-formed Haskell expression (typechecking is done later). A $\langle chain\rangle$ is a list of same-typed Haskell expressions that are intercalated by $\langle expln\rangle$s (which can be represented as an initial term and a list of $\langle clause\rangle$s, where each $\langle clause\rangle$ is a term along with an explanation of how it is liquid-equal (via EqualProp) to the previous term. An $\langle expln\rangle$ is an explanation of the liquid equality (EqualProp) of two adjacent terms in a $\langle chain\rangle$.

$$
\begin{array}{rcl}
\langle chain\rangle & ::= & (\langle expr\rangle, [\langle clause\rangle]) \\
\langle clause\rangle & ::= & (\langle expr\rangle, \langle expln\rangle) \\
\langle expln\rangle & ::= & \text{by trivial} \\
& | & \text{by } \langle proof\rangle \\
& | & \text{by reflexivity} \\
& | & \text{by symmetry } \langle expln\rangle \\
& | & \text{by rewrite } \langle expr\rangle \text{ to } \langle expr\rangle\ \langle expln\rangle \\
& | & \text{by extend } \langle pattern\rangle\ \langle expln\rangle \\
& | & \text{by smt by } \langle expr\rangle
\end{array}
$$

The functions go and reify are defined in a scope nested under lift's scope. Note the convention that variable terms are substituted underneath quotes.

$$
\begin{array}{rcl}
\text{lift} & : & \langle clause\rangle \to \langle\text{“}expr\text{”}\rangle \\
\text{lift } (t_1, cs) & = & \text{go (reverse } cs)
\end{array}
$$

$$
\begin{array}{rcll}
\text{go} & : & [\langle clause\rangle] \to \langle\text{“}expr\text{”}\rangle & \\
\text{go } [] & = & \text{“reflexivity } t_1\text{”} & \\
\text{go } ((t_2, e) : []) & = & \text{“transitivity } t_1\ t_2\ t_2\ p_{12}\ (\text{reflexivity } t_2)\text{”} & \textbf{where } p_{12} \leftarrow \text{reify } t_1\ t_2\ e \\
\text{go } ((t_3, e_{23}) : (t_2, e_{12}) : []) & = & \text{“transitivity } t_1\ t_2\ t_3\ p_{12}\ p_{23}\text{”} & \textbf{where } p_{12} \leftarrow \text{reify } t_1\ t_2\ e_{12};\ p_{23} \leftarrow \text{reify } t_2\ t_3\ e_{23} \\
\text{go } ((t_k, e_{jk}) : (t_j, e_{ij}) : cs) & = & \text{“transitivity } t_1\ t_j\ t_k\ e_{1j}\ e_{jk}\text{)”} & \textbf{where } e_{1j} \leftarrow \text{go } ((t_j, e_{ij}) : cs);\ e_{jk} \leftarrow \text{reify } t_j\ t_k\ e_{jk}
\end{array}
$$

$$
\begin{array}{rcll}
\text{reify} & : & \langle expr\rangle \to \langle expr\rangle \to \langle expln\rangle \to \langle\text{“}expr\text{”}\rangle & \\
\text{reify } t_i\ t_j\ (\text{by trivial}) & = & \text{“reflexivity } t_i\text{”} & \\
\text{reify } t_i\ t_j\ (\text{by } p_{ij}) & = & \text{“}p_{ij}\text{”} & \\
\text{reify } t_i\ t_j\ (\text{by reflexivity}) & = & \text{“reflexivity } t_i\text{”} & \\
\text{reify } t_i\ t_j\ (\text{by symmetry } e) & = & \text{“symmetry} t_j t_i p_{ij}\text{”} & \textbf{where } e_{ij} \leftarrow \text{reify } t_i\ t_j\ e \\
\text{reify } t_i\ t_j\ (\text{by rewrite } t_3 \text{ to } t_4\ e) & = & \text{rewrite “}t_3\text{” “}t_2\text{” “}p_{12}\text{” “}t_i\text{”} & \textbf{where } e_{12} \leftarrow \text{reify } t_3\ t_4\ e \\
\text{reify } t_i\ t_j\ (\text{by extend } pat\ e) & = & \text{“extensionality } t_i\ t_j\ p_{ij}\text{”} & \textbf{where } p'_{ij} \leftarrow \text{reify } (t_i\ p)\ (t_j\ p)\ e;\ p_{ij} \leftarrow \text{“}\lambda\ p \to p'_{ij}?\ t_i\ p\ ?\ t_j\ p\text{”} \\
\text{reify } t_i\ t_j\ (\text{by smt by } p_{ij}) & = & \text{“reflexivity } t_i\ ?\ p_{ij}\text{”} &
\end{array}
$$