4.4 Formulating SMT Statements

So far we have described the inference rules and the subtyping rule. We have yet to algorithm that can tying a valid type for a set of given subtyping rules.

Definition 4.1: Template

We say $\hat{T}$ is a *template* $:\Leftrightarrow$

$$\begin{aligned} & T \text{ is of form } \{\nu \in \mathit{Int} | [\kappa_i]_S\} \\ & \text{where } i \in \mathbb{N} \text{and} S : \mathcal{V} \nrightarrow \mathcal{Q} \\ \vee\ & T \text{ is of form } a : \{\nu \in \mathit{Int} | [\kappa_i]_S\} \to \hat{T} \\ & \text{where } i \in \mathbb{N}, \hat{T} \text{ is a template and} S : \mathcal{V} \nrightarrow \mathcal{Q}. \end{aligned}$$

we call κ_i for $i \in \mathcal{Q}$ a *liquid type variable.*

A Template will be used for a liquid type with unknown refinement. Note that the inference rule for function application introduces a refinement substitution S. For template this substitution is not defined and needs to be delaying until after the corresponding liquid type has been derived.

To transform a template into a liquid type, we need to substitute all liquid type variables with refinements. For this the term-wise substitution will be used.

Our algorithm will resolve a set of suptying condition for templates:

Definition 4.2: Condition

Let $\Theta : \mathcal{V} \nrightarrow \mathcal{T}$ and $\Lambda \subset \mathcal{Q}$.

—

We say c is a *Condition* $:\Leftrightarrow$ c is of form $\hat{T}_1 <:_{\Theta,\Lambda} \hat{T}_2$ where $\hat{T}_1, \hat{T}_2$ are templates.

We will also need a function to obtain the set of all liquid type variables of a template or condition.

Definition 4.3: Vars

Given a template $\hat{T}$, we define $\text{Vars}(\hat{T})$ as follows.

$$\begin{aligned} \text{Vars}(\{\nu \in \mathit{Int} | \kappa_i\}) =& \{\kappa_i\} \\ \text{Vars}(a : \{\nu \in \mathit{Int} | \kappa_i\} \to \hat{T}) =& \{\kappa_i\} \cup \text{Vars}(\hat{T}) \end{aligned}$$

Given a condition c, we define $\text{Vars}(c)$ as follows.

$$\text{Vars}(\hat{T}_1 <:_{\Theta,\Lambda} \hat{T}_2) = \text{Vars}(\hat{T}_1) \cup \text{Vars}(\hat{T}_2) \cup \{\text{Vars}(\hat{T}_3) | (\_, \hat{T}_3) \in \Theta\}$$

The main idea of the algorithm is to first generate a set of predicates and then exclude elements of it until all conditions are valid for the remaining predicates. By conjunction over all remaining predicates we result in a valid refinement.

We therefore need a function, depending on a set of variable $\mathcal{Q}$, that will generate a set of predicates. Note that the resulting set should be finite and a subset of. If the generated set is to small, then our resulting conditions might be to weak.

$$
\begin{aligned}
\mathit{Init} : \mathcal{P}(\mathcal{V}) \to & \mathcal{P}(\mathcal{Q}) \\
\mathit{Init}(Q) ::= & 0 < \nu \\
& \mid Q < \nu \\
& \mid \nu < 0 \\
& \mid \nu < Q \\
& \mid \nu = Q \\
& \mid \nu = 0 \\
& \mid \neg(\nu = Q) \\
& \mid \neg(\nu = 0)
\end{aligned}
$$

We can always extend the realm of predicates if the resulting refinements are too weak.

4.4.1 The Inference Algorithm

As an input we require a set of conditions C and $\Theta : \mathcal{V} \nrightarrow \mathcal{T}$. The result will be a valid liquid type.

$$
\begin{aligned}
\text{Infer}(\Theta, C) = \text{let } & I := \{(\kappa, \mathit{Init}(\{a \mid (a, \_) \in \Theta\})) \mid \kappa \in \bigcup_{c \in C} \text{Var}(c)\} \\
& A := \text{Solve}(\Theta, \bigcup_{c \in C} \text{Split}(c), I), \\
\text{in } & \{(\kappa, \bigwedge Q) \mid (\kappa, Q) \in A\}
\end{aligned}
$$

We start by spliting the conditions for functions into conditions for simpler liquid types.

$$
\begin{aligned}
\text{Split}(a : \hat{T}_1 \to \hat{T}_2 <:_{\Theta,\Lambda} a : \hat{T}_3 \to \hat{T}_4) =& \{\hat{T}_3 <:_{\Theta,\Lambda} \hat{T}_1, \hat{T}_2 <:_{\Theta \cup \{(a,\hat{T}_3)\},\Lambda} \hat{T}_4\} \\
\text{Split}(\{\nu : \mathit{Int} | r_1\} <:_{\Theta,\Lambda} \{\nu : \mathit{Int} | r_2\}) =& \{\{\nu : \mathit{Int} | r_1\} <:_{\Theta,\Lambda} \{\nu : \mathit{Int} | r_2\}\}
\end{aligned}
$$

We resolve the obtained conditions by repeatably checking if a condition is not valid and removing all predicates that contradict it. By removing the predicate we weaken the resulting refinement.

$$
\begin{aligned}
&\text{Solve}(\Theta, C, A) = \\
&\qquad \text{let } S := \{(\kappa, \bigwedge Q) \mid (\kappa, Q) \in A\} \\
&\qquad \text{in } \begin{cases} \text{Solve}(C, \text{Weaken}(c, A)) & \text{if } c \in C \text{ exists, such that } [\![[c]_S]\!]_\Theta \text{ is not valid} \\ A & \text{otherwise} \end{cases}
\end{aligned}
$$

Note that we can use a SMT solver to validate $[\![[c]_S]\!]_\Theta$

$$
\begin{aligned}
&\mathrm{Weaken}(\Lambda, \{\nu : Int | r\} <:_{\Theta,\Lambda} \{\nu : Int | [\kappa_0]_{S_0}\}, A) = \\
&\quad \text{let } S := \{(\kappa, \bigwedge Q) \mid (\kappa, Q) \in A\}, \\
&\qquad p := \bigwedge \{[q]_S \mid q \in \Lambda\}, \\
&\qquad Q_0 := \{q \mid q \in A(\kappa_0) \wedge ([\![p \wedge r]\!]_\Theta \Rightarrow [\![[r_2]_{S_0}]\!]_\Theta)\} \\
&\quad \text{in } \{(\kappa, Q) \mid (\kappa, Q) \in A \wedge \kappa \neq \kappa_0\} \cup \{(\kappa_0 | Q_0)\}
\end{aligned}
$$

Note that we can use a SMT solver to validate $[\![p \wedge r]\!]_\Theta \Rightarrow [\![[r_2]_{S_0}]\!]_\Theta$.