1. Type Safety via Logical Relations

We sketch a proof of type-safety of the DOT calculus via step-indexed logical relations [1–3].

1.1 Type Safety

Type-safety states that a well-typed program doesn't get stuck. More formally: If $\emptyset \vdash t : T$ and $t \mid \emptyset \rightarrow^* t' \mid s'$ then either t' is a value or $\exists t'', s''. t' \mid s' \rightarrow t'' \mid s''$.

Our strategy is to define a logical relation $\Gamma \vDash t : T$, such that $\Gamma \vdash t : T$ implies $\Gamma \vDash t : T$ implies type-safety.

1.2 Step-Indexed Logical Relations

In order to ensure that our logical relation is well-founded, we use a step index. For each step index k, we define the set of values and the set of terms that appear to belong to a given type, when taking at most k steps. $\Gamma \vDash t : T$ is then defined in terms of the step-indexed logical relation by requiring it to hold $\forall k$.

1.2.1 Set of Values

$\mathcal{V}_{k;\Gamma;s}\llbracket T \rrbracket$ defines the set of values that appear to have type T when taking at most k steps. Γ and s must agree: $dom(\Gamma) = dom(s)$ (ordered) and $\forall (x : T) \in \Gamma, x \in \mathcal{V}_{k;\Gamma;s}\llbracket T \rrbracket$. A variable y belongs to $\mathcal{V}_{0;\Gamma;s}\llbracket T \rrbracket$ simply by being in the store. In addition, it belongs to $\mathcal{V}_{k;\Gamma;s}\llbracket T \rrbracket$ for $k > 0$, if it defines all type, method and value labels in the expansion of T appropriately for $j < k$ steps.

$$
\begin{aligned}
&\mathcal{V}_{k;\Gamma;s}\llbracket T \rrbracket = \{ y \mid y \in dom(s) \ \wedge \ (\\
&(\Gamma \vdash T \ \mathbf{wfe} \ \wedge \\
&\quad \forall j < k, \\
&\quad y \mapsto T_c \left\{ \overline{l = v} \ \overline{m(x) = t} \right\} \in s, \\
&\quad \Gamma \vdash T \prec_y \overline{D}, \\
&\quad (\forall L_i : S \to U \in \overline{D}, \\
&\qquad \Gamma \vdash y \ni L_i : S'..U') \ \wedge \\
&\quad (\forall m_i : S \to U \in \overline{D}, \\
&\qquad t_i \in \mathcal{E}_{j;\Gamma, x_i : S; s}\llbracket U \rrbracket) \ \wedge \\
&\quad (\forall l_i : V \in \overline{D}, \\
&\qquad v_i \in \mathcal{V}_{j;\Gamma;s}\llbracket V \rrbracket)) \vee \\
&(T = T_1 \wedge T_2 \ \wedge \ y \in \mathcal{V}_{k;\Gamma;s}\llbracket T_1 \rrbracket \ \wedge \ y \in \mathcal{V}_{k;\Gamma;s}\llbracket T_2 \rrbracket) \vee \\
&(T = T_1 \vee T_2 \ \wedge \ (y \in \mathcal{V}_{k;\Gamma;s}\llbracket T_1 \rrbracket \vee y \in \mathcal{V}_{k;\Gamma;s}\llbracket T_2 \rrbracket)) \\
&)\}
\end{aligned}
$$

This relation captures the observation that the only ways for a term to get stuck is to have a field selection on an uninitialized field or a method invocation on an uninitialized method. However, a *potential pitfall* is that the value itself might occur in the types S, U, V, because we substitute it for the "self" occurrences in the expansion, so the relation makes sure that the required type labels exist.

1.2.2 Set of Terms

$\mathcal{E}_{k;\Gamma;s}\llbracket T \rrbracket$ defines the set of terms that appear to have type T when taking at most k steps. s must agree with a *prefix* of Γ, so Γ can additionally contain variables not in s. This is needed for checking methods in $\mathcal{V}$ above, and for relating open terms. If $k > 0$, $\mathcal{E}$ extends Γ and s so that they agree. It then states that if it can reduce t in the extended store to an irreducible term in $j < k$ steps, then this term must be in a corresponding $\mathcal{V}$ set with Γ now extended to agree with the store resulting from the reduction steps.

irred (t, s) is a shorthand for $\neg \exists t', s'. t \mid s \rightarrow t' \mid s'$. $\supseteq$ is used initially for the possibly shorter store to agree with the environment, and can extend both in many different ways. $\supseteq^!$ is used finally for the possibly shorter environment to agree with the store, and just extends the environment in one straightforward way: hence, it defines singleton sets.

$$
\begin{aligned}
&\mathcal{E}_{k;\Gamma;s}\llbracket T \rrbracket = \{ t \mid \\
&\quad k = 0 \vee (\forall j < k, \\
&\qquad \forall (\Gamma'; s') \in \supseteq_k \llbracket \Gamma; s \rrbracket, \\
&\qquad t \mid s' \rightarrow^j t' \mid s'' \ \wedge \\
&\qquad \text{irred } (t', s'') \rightarrow \\
&\qquad \forall \Gamma'' \in {\supseteq^!}_{k;s''}\llbracket \Gamma' \rrbracket, \\
&\qquad t' \in \mathcal{V}_{k-j-1;\Gamma'';s''}\llbracket T \rrbracket) \\
&\}
\end{aligned}
$$

1.2.3 Extending the environment and the store

$\supseteq_k \llbracket \Gamma; s \rrbracket$ for $k > 0$ defines the set of completed environment and stores that agree on $k - 1$ steps, and that extend Γ and s. s must agree with a *prefix* of Γ. Both Γ and s are ordered maps. For s, s' extends s if s is a prefix of s'. For Γ, Γ' extends Γ if we get back Γ by keeping only the elements of Γ' that belong to Γ. Furthermore, a prefix of Γ' agrees with s.

$$
\begin{aligned}
&\supseteq_k \llbracket \Gamma; s \rrbracket = \{ \\
&(\overline{x : T}^m, \overline{x_{ij} : T_{ij}}^{m \le i < n; 0 \le j < i_n}; s, \overline{x_{ij} \mapsto c_{ij}}^{m \le i < n; 0 \le j < i_n}) \mid \\
&s = \overline{x \mapsto c}^m \ \wedge \ \Gamma = \overline{x : T}^n \ \wedge \\
&m \le n \ \wedge \ \forall i, m \le i < n, \forall i_n, j, 0 \le j < i_n, \\
&\forall T_{ij}, c_{ij}, T_{i(i_n - 1)} = T_i, \forall n' \le n, i_{n'} \le i_n, \\
&c_{ij} \in \mathcal{V}_{k-1; \overline{x:T}^m, \overline{x_{ij}:T_{ij}}^{m \le i < n'; 0 \le j < i_{n'}}; s, \overline{x_{ij} \mapsto c_{ij}}^{m \le i < n'; 0 \le j < i_{n'}}} \llbracket T_{ij} \rrbracket \\
&\}
\end{aligned}
$$

1.2.4 Completing the environment to agree with the store

${\supseteq^!}_{k;s}\llbracket \Gamma \rrbracket$ defines a singleton set of a completed environment that agrees with a store s by simply copying the constructor type from the store for each missing variable.

$$
\begin{aligned}
&{\supseteq^!}_{k;s}\llbracket \Gamma \rrbracket = \{ \Gamma, \overline{x_i : T_{c_i}}^{m \le i < n} \mid \\
&\quad \Gamma = \overline{x : T}^m \ \wedge \ s = \overline{x \mapsto c}^n \\
&\}
\end{aligned}
$$

1.2.5 Terms in the Logical Relation

$\Gamma \vDash t : T$ is simply defined as $t \in \mathcal{E}_{k;\Gamma;\emptyset}\llbracket T \rrbracket, \forall k$.

1.3 Statements and Proofs

1.3.1 Fundamental Theorem

The fundamental theorem is the implication from $\Gamma \vdash t : T$ to $\Gamma \vDash t : T$. Type safety is a straightforward corollary of this theorem.

Proof: The proof is on induction on the derivation of $\Gamma \vdash t : T$. For each case, we need to show $t \in \mathcal{E}_{k;\Gamma;\emptyset}\llbracket T \rrbracket, \forall k$. The non-trivial case is when $k > 0$ and for $(\Gamma'; s') \in \supseteq_k \llbracket \Gamma; s \rrbracket$ and some $j < k$, $t \mid s \rightarrow^j t' \mid s' \ \wedge$ irred (t', s'). Then, we need to show $t' \in \mathcal{V}_{k-j-1;\Gamma'';s'}\llbracket T \rrbracket$ for $\Gamma'' \in {\supseteq^!}_{\Gamma;k}\llbracket s' \rrbracket \Gamma'$.

Case VAR: $\Gamma \vdash x : T$ knowing $(x : T) \in \Gamma$. $x \in \mathcal{V}_{k-1;\Gamma';s}\llbracket T \rrbracket$ follows from the definition of $\supseteq_k \llbracket \Gamma; \emptyset \rrbracket$.

Case SEL: $\Gamma \vdash t_1.l_i : T$ knowing $\Gamma \vdash t_1 : T_1$, $\Gamma \vdash T_1 \prec_z \overline{D}$, $l_i : V_i \in \overline{D}$ and knowing either that $t_1 = p_1 \wedge T = [p/z]V_i$ or that $z \notin fn(V_i) \wedge T = V_i$.

By operational semantics and induction hypothesis, $t_1 \mid s \rightarrow^{j-1} t_1' \mid s'$ and irred (t_1', s') and $t_1' \in \mathcal{V}_{k-j+1-1;\Gamma';s'}[\![T_1]\!]$.

By operational semantics and the above, $t_1'.l_i \mid s' \rightarrow^1 t' \mid s'$, and we can conclude $t' \in \mathcal{V}_{k-j-1;\Gamma'';s'}[\![T]\!]$ from the clause for value labels of $t_1' \in \mathcal{V}_{k-j;\Gamma'';s'}[\![T_1]\!]$.

Case MSEL: $\Gamma \vdash t_1.m_i(t_2) : T$ knowing $\Gamma \vdash t_1 : T_1$, $\Gamma \vdash t_2 : T_2$, $\Gamma \vdash T_1 \prec_z \overline{D}$, $m_i : S_i \rightarrow U_i \in \overline{D}$ and knowing either that $t_1 = p_1 \wedge S = [p/z]S_i \wedge T = [p/z]U_i$ or that $z \notin fn(S_i) \wedge z \notin fn(U_i) \wedge S = S_i \wedge T = U_i$, and knowing that $\Gamma \vdash T_2 <: S$.

By operational semantics and induction hypotheses, $t_1 \mid s \rightarrow^{j_1} t_1' \mid s_1$ and irred (t_1', s_1) and $t_2 \mid s \rightarrow^{j_2} t_2' \mid s_2$ and irred (t_2', s_2) and $t_1' \in \mathcal{V}_{k-j_1-1;\Gamma_1;s_1}[\![T_1]\!]$ and $t_2' \in \mathcal{V}_{k-j_2-1;\Gamma_2;s_2}[\![T_2]\!]$.

Because t_2 reduces to a value t_2' starting in store s, it should also reduce to a value v_2 in the same number of steps starting in store s_1, since s_1 extends s. So let $t_2 \mid s_1 \rightarrow^{j_2} v_2 \mid s_{12}$ with $v_2 \in \mathcal{V}_{k-j_2-1;\Gamma_{12};s_{12}}[\![T_2]\!]$.

By the above and operational semantics, $t_1'.m_i(v_2) \mid s_{12} \rightarrow^1 [v_2/x_i]t_i \mid s_{12}$.

By the substitution lemma, $[v_2/x_i]t_i \in \mathcal{E}_{k-\max(j_1,j_2)-1;\Gamma_{12};s_{12}}[\![T]\!]$. Supposing, $[v_2/x_i]t_i \mid s_{12} \rightarrow^{j_3} t' \mid s'$, with $j_1 + j_2 + j_3 + 1 = j$, this completes the case, by monotonicity of $\mathcal{V}$.

Case NEW: $\Gamma \vdash$ **val** $y =$ **new** $c;\ t_b : T$ knowing ...

By operational semantics, **val** $y =$ **new** $c;\ t_b \mid s \rightarrow^1 t_b \mid s_b$ where $s_b = s, y \mapsto c$. So $t_b \mid s_b \rightarrow^{j-1} t' \mid s'$.

By induction hypotheses, $y \in \mathcal{V}_{k;\Gamma_b;s_b}[\![T_c]\!]$ and $t_b \in \mathcal{E}_{k;\Gamma_b;s_b}[\![T]\!]$.

Result follows by monotonicity of $\mathcal{V}$.

□

1.3.2 Substitution Lemma

The substitution lemma states that if (1) $v \in \mathcal{V}_{k_2;\Gamma_{12};s_{12}}[\![T_2]\!]$ and (2) $t \in \mathcal{E}_{k_1;\Gamma_1,x:S;s_1}[\![T]\!]$ and (3) $\Gamma \vdash T_2 <: S$ with (4) $x \notin fn(T)$ and Γ_1 extends Γ and Γ_{12} extends Γ_1 and s_{12} extends s_1 and Γ_1 agrees with s_1 and Γ_{12} agrees with s_{12} and a prefix of Γ_{12} agrees with s_1, then $[v/x]t \in \mathcal{E}_{\min(k_1,k_2);\Gamma_{12};s_{12}}[\![T]\!]$.

Proof Sketch: By (1) and (3), it should hold that (5) $v \in \mathcal{V}_{k_2;\Gamma_{12};s_{12}}[\![S]\!]$ by the subset semantics lemma. Since (2) holds, it should also hold that $t \in \mathcal{E}_{\min(k_1,k_2);\Gamma_{12},x:S;s_{12}}[\![T]\!]$ by the extended monotonicity lemma. Then, we can instantiate x in the complete store to map to what v maps to. This should be fine by (5) and monotonicity. Thus, $t \in \mathcal{E}_{\min(k_1,k_2);\Gamma_{12},x:S;s_{12},x\mapsto s_{12}(v)}[\![T]\!]$. Thanks to (4), we don't actually need x to be held abstract in the environment, because it won't occur in T or its expansion (a *potential pitfall* is whether its occurrences in t_i could still cause a check to fail through narrowing issues), so we can use the type of v in the environment instead of S for x: $t \in \mathcal{E}_{\min(k_1,k_2);\Gamma_{12},x:\Gamma_{12}(v);s_{12},x\mapsto s_{12}(v)}[\![T]\!]$. This implies what needs to be shown. □

1.3.3 Subset Semantics Lemma

The subset semantics lemma states that if $v \in \mathcal{V}_{k;\Gamma;s}[\![S]\!]$ and $\Gamma \vdash S <: U$, then $v \in \mathcal{V}_{k;\Gamma;s}[\![U]\!]$.

Proof Sketch: Because S is a subtype of U, it should hold that the expansion of S subsumes the expansion of U, when the "self" occurrences are of type S. Therefore, for $v \in \mathcal{V}_{k;\Gamma;s}[\![U]\!]$, we have fewer declarations to check than for $v \in \mathcal{V}_{k;\Gamma;s}[\![S]\!]$.

A *potential pitfall* is whether some types of the expansion of U can become non-expanding when the "self" occurrences are actually v instead of just abstractly of type S, causing a check to fail. Another worry is that such a non-expanding type results from narrowing of a parameter type. □

1.3.4 Extended Monotonicity Lemma

The extended monotonicity lemma states that if $t \in \mathcal{E}_{k;\Gamma,x:S;s}[\![T]\!]$ then $t \in \mathcal{E}_{j;\Gamma',x:S;s'}[\![T]\!]$ for $j \leq k$, Γ' extends Γ, s' extends s, and Γ agrees with s and a prefix of Γ' agrees with s.

Proof Sketch: For the monotonicity with regards to the step index, this follows directly from the definitions of $\mathcal{E}$ and $\mathcal{V}$. For the environment and the store, this follows by design from the definition of $\supseteq_k[\![\Gamma, x : S; s]\!]$. To extend the environment and the store for $x : S$, we can append as much as we want to Γ and s, to get Γ' and s', and then ignore the last element which is for $x : S$. □

References

[1] A. J. Ahmed. *Semantics of types for mutable state*. PhD thesis, Princeton University, 2004.

[2] A. J. Ahmed. Step-indexed syntactic logical relations for recursive and quantified types. In *ESOP*, pages 69–83, 2006.

[3] C. Hritcu and J. Schwinghammer. A step-indexed semantics of imperative objects. *Logical Methods in Computer Science*, 5(4), 2009.