A SUFFICIENT CONDITIONS FOR CANONICALIZATION FUNCTIONS

We found the following theory helpful for thinking about and working with canonicalization functions.

Definition A.1 (Canonicalization function ingredients). Let $U \subseteq \text{Prog}_{\mathcal{L}^?} \times \text{Prog}_{\mathcal{L}^?} \times \Sigma_{\mathcal{L}}$ be a relation and $\phi : \text{Prog}_{\mathcal{L}^?} \to \text{Prog}_{\mathcal{L}^?}$ a function. Then:

- ϕ is **substitution-preserving on** $[\![\cdot]\!]_{\mathcal{L}}$ if for all $S \in \text{Prog}_{\mathcal{L}^?}$ and $\sigma \in \Sigma_{\mathcal{L}}$, $[\![\phi(\sigma S)]\!]_{\mathcal{L}} = [\![\sigma\phi(S)]\!]_{\mathcal{L}}$.
- ϕ is **injective on** $[\![\cdot]\!]_{\mathcal{L}}$ if for all $P_1, P_2 \in \text{Prog}_{\mathcal{L}}$, $[\![\phi(P_1)]\!]_{\mathcal{L}} = [\![\phi(P_2)]\!]_{\mathcal{L}} \Rightarrow [\![P_1]\!]_{\mathcal{L}} = [\![P_2]\!]_{\mathcal{L}}$.
- ϕ is **semantics-preserving** if for all $S \in \text{Prog}_{\mathcal{L}^?}$, $[\![\phi(S)]\!]_{\mathcal{L}^?} = [\![S]\!]_{\mathcal{L}^?}$.

LEMMA A.2 (CANONICALIZATION FUNCTION RECIPE). *Let U be a partial semantic unification relation for $\mathcal{L}$ and $\phi : Prog_{\mathcal{L}^?} \to Prog_{\mathcal{L}^?}$ be a computable function that is substitution-preserving on $[\![\cdot]\!]_{\mathcal{L}}$ and injective on $[\![\cdot]\!]_{\mathcal{L}}$. Then ϕ is a canonicalization function for $\mathcal{L}$ with respect to U.*

PROOF. We have

$$\begin{aligned}
(S_1, S_2, \sigma) \in \text{Ker}_U\, \phi &\Rightarrow U(\phi(S_1), \phi(S_2), \sigma) \\
&\Rightarrow [\![\sigma\phi(S_1)]\!]_{\mathcal{L}} = [\![\sigma\phi(S_2)]\!]_{\mathcal{L}} \\
&\Rightarrow [\![\phi(\sigma S_1)]\!]_{\mathcal{L}} = [\![\phi(\sigma S_2)]\!]_{\mathcal{L}} \\
&\Rightarrow [\![\sigma S_1]\!]_{\mathcal{L}} = [\![\sigma S_2]\!]_{\mathcal{L}} \\
&\Rightarrow [\![S_1]\!] \equiv^{\sigma}_{\mathcal{L}} [\![S_2]\!] \\
&\Rightarrow (S_1, S_2, \sigma) \in \text{Ker}_{\equiv_{\mathcal{L}}} [\![\cdot]\!]_{\mathcal{L}^?},
\end{aligned}$$

so $\text{Ker}_U\, \phi \leq \text{Ker}_{\equiv_{\mathcal{L}}} [\![\cdot]\!]_{\mathcal{L}^?}$. □

LEMMA A.3. *If $\phi : Prog_{\mathcal{L}^?} \to Prog_{\mathcal{L}^?}$ is semantics-preserving, then ϕ is (i) substitution-preserving on $[\![\cdot]\!]_{\mathcal{L}}$ and (ii) injective on $[\![\cdot]\!]_{\mathcal{L}}$.*

PROOF. For (i), if $S \in \text{Prog}_{\mathcal{L}^?}$ and $\sigma \in \Sigma_{\mathcal{L}}$, then

$$[\![\phi(\sigma S)]\!]_{\mathcal{L}} = [\![\sigma S]\!]_{\mathcal{L}} = [\![S]\!]_{\mathcal{L}^?}(\sigma) = [\![\phi(S)]\!]_{\mathcal{L}^?}(\sigma) = [\![\sigma\phi(S)]\!]_{\mathcal{L}}.$$

For (ii), the result is immediate. □

B PROOFS FOR MAIN PAPER

We now provide proofs of the lemmas and theorem in the main paper.

Theorem 4.5. *Algorithm 1 always terminates, and if it returns $\top$ on (ϕ, P_1, P_2), then $[\![P_1]\!] = [\![P_2]\!]$.*

Proof. Termination is immediate from the computability of canonicalization functions and the decidable syntactic equality of Prog. Correctness is immediate from the fact that $\mathrm{Ker}\,\phi \leq \mathrm{Ker}\,[\![\cdot]\!]$. □

Lemma 4.13. *Let U be a partial semantic unification relation for $\mathcal{L}$ and $\phi : \mathit{Prog}_{\mathcal{L}^?} \to \mathit{Prog}_{\mathcal{L}^?}$ be computable and semantics-preserving. Then ϕ is a canonicalization function for $\mathcal{L}$ with respect to U.*

Proof. Immediate corollary of Lemma A.2 and Lemma A.3. □

Theorem 4.15. *Let ϕ be a canonicalization function for a hole-free language $\mathcal{L}$ with respect to U and u be an inference algorithm for U. Then $\textsc{Ebc}(\phi, u, \cdot, \cdot)$ is a semi-inference algorithm for $\mathrm{Ker}_{\equiv_\mathcal{L}}\,[\![\cdot]\!]_{\mathcal{L}^?}$.*

Proof. Computability follows from the computability of ϕ and u. For correctness, suppose $\textsc{Ebc}(\phi, u, S_1, S_2) = \sigma$. Then $u(\phi(S_1), \phi(S_2)) = \sigma$, so $U(\phi(S_1), \phi(S_2), \sigma)$. As $\mathrm{Ker}_U\,\phi \leq \mathrm{Ker}_{\equiv_\mathcal{L}}\,[\![\cdot]\!]_{\mathcal{L}^?}$, we have $[\![S_1]\!]_{\mathcal{L}^?} \equiv^\sigma_\mathcal{L} [\![S_2]\!]_{\mathcal{L}^?}$. □

Theorem 4.19. *Let ϕ_1 and ϕ_2 be canonicalization functions for a hole-free language $\mathcal{L}$ with respect to U_1 and U_2 and let u_1 and u_2 be inference algorithms for U_1 and U_2. Suppose $\phi_1 \geq \phi_2$. Then:*

1. *If $\textsc{Ebc}(\phi_2, u_2, S_1, S_2) = \sigma_2$ for some hole substitution $\sigma_2 \in \Sigma$, then $\textsc{Ebc}(\phi_1, u_1, S_1, S_2) = \sigma_1$ for some hole substitution $\sigma_1 \in \Sigma$.*
2. *If $\phi_1 > \phi_2$, there exist $S_1, S_2 \in \mathrm{cl}_?\,\mathit{Prog}$ with $\textsc{Ebc}(\phi_2, u_2, S_1, S_2) = \bot$ yet $\textsc{Ebc}(\phi_1, u_1, S_1, S_2) = \sigma_1$ for some hole substitution $\sigma_1 \in \Sigma$.*

Proof.

1. We must have $u_2(\phi_2(S_1), \phi_2(S_2)) = \sigma_2$, so $U_2(\phi_2(S_1), \phi_2(S_2), \sigma_2)$. Therefore,
$$(S_1, S_2) \in \mathcal{F}(\mathrm{Ker}_{U_2}\,\phi_2) \leq \mathcal{F}(\mathrm{Ker}_{U_1}\,\phi_1),$$
so there exists some $\sigma_1 \in \Sigma$ such that $U_1(\phi_1(S_1), \phi_1(S_1), \sigma_1)$; the result follows from the fact that u_1 is an inference algorithm.
2. We can take any $(S_1, S_2) \in \mathcal{F}(\mathrm{Ker}_{U_1}\,\phi_1) \setminus \mathcal{F}(\mathrm{Ker}_{U_2}\,\phi_2)$, which must be nonempty because the refinement is strict. □

Theorem 5.3. *Let ϕ be a canonicalization function for a hole-free language $\mathcal{L}$ with respect to U, u be an inference algorithm for U, $\textsc{Enum}$ be an enumerator $\mathit{Lib}_\mathcal{L}$, and P be a program in $\mathit{Prog}_\mathcal{L}$. If Algorithm 3 terminates, it returns a component sketch $S \in \mathrm{cl}_?\,\mathit{Lib}_\mathcal{L}$ and hole substitution σ such that $[\![\sigma S]\!]_\mathcal{L} = [\![P]\!]_\mathcal{L}$.*

Proof. If Algorithm 3 returns (S, σ), then $\textsc{Ebc}(\phi, u, P, S) = \sigma$, so we have $[\![S]\!]_{\mathcal{L}^?} \equiv^\sigma_\mathcal{L} [\![P]\!]_{\mathcal{L}^?}$ by Theorem 4.15. Hence, $[\![\sigma S]\!] = [\![\sigma P]\!] = [\![P]\!]$, where the second equality holds because $\mathcal{L}$ is hole-free. □

C COBBLER'S LIBRARY COMPONENTS

We chose the components to include in Cobbler's library based on Cobbler s empirical performance on the training set (and not the test set).

For CBR-Elm, we include the following 17 components mostly from the standard library:[5]

(1) `Basics.or` (`||`)
(2) `Basics.and` (`&&`)
(3) `Basics.not`

(4) `Maybe.map`
(5) `Maybe.withDefault`

(6) `Result.map`
(7) `Result.mapError`
(8) `Result.withDefault`

(9) `List.append`
(10) `List.map`
(11) `List.filter`
(12) `List.concat`
(13) `List.any`
(14) `List.head`
(15) `List.tail`
(16) `List.find` (not present in Elm standard library)
(17) `List.findMap` (not present in Elm standard library)

For CBR-Python, we include the following 21 NumPy components:

(1) `np.sum`
(2) `np.prod`

(3) Filtering (e.g. `x[x > 0]`)

(4) `np.multiply`
(5) `np.divide`
(6) `np.add`
(7) `np.subtract`
(8) `np.power`
(9) `np.equal`
(10) `np.not_equal`
(11) `np.full`
(12) `np.greater`
(13) `np.greater_equal`
(14) `np.less`
(15) `np.less_equal`
(16) `np.where`
(17) `np.roll`
(18) `np.convolve` (with "valid" option)
(19) `np.random.randint` (with "size" argument)
(20) `np.arange`
(21) `np.copy`

As mentioned in Section 6.2, we also include two additional functions for CBR-Python we call *cosmetic* (`list` and `np.vectorize`), which do not provide any performance benefits but can expose opportunities to apply other functions.

[5] https://package.elm-lang.org/packages/elm/core/latest/