

Francesco Mecca

1 Correctness of the algorithm

Running a program t_S or its translation $\llbracket t_S \rrbracket$ against an input v_S produces as a result r in the following way:

$$\begin{aligned} (\llbracket t_S \rrbracket_S(v_S) = C_S(v_S)) &\rightarrow r \\ t_S(v_S) &\rightarrow r \end{aligned}$$

Likewise

$$\begin{aligned} (\llbracket t_T \rrbracket_T(v_T) = C_T(v_T)) &\rightarrow r \\ t_T(v_T) &\rightarrow r \end{aligned}$$

where result $r ::= \text{guard list} * (\text{Match blackbox} \mid \text{NoMatch} \mid \text{Absurd})$ and guard $::= \text{blackbox}$.

Having defined equivalence between two inputs of which one is expressed in the source language and the other in the target language $v_S \simeq v_T$ (TODO define, this talks about the representation of source values in the target)

we can define the equivalence between a couple of programs or a couple of decision trees

$$\begin{aligned} t_S \simeq t_T &:= \forall v_S \simeq v_T, t_S(v_S) = t_T(v_T) \\ C_S \simeq C_T &:= \forall v_S \simeq v_T, C_S(v_S) = C_T(v_T) \end{aligned}$$

The proposed equivalence algorithm that works on a couple of decision trees is returns either *Yes* or *No*(v_S, v_T) where v_S and v_T are a couple of possible counter examples for which the constraint trees produce a different result.

1.1 Statements

Theorem. We say that a translation of a source program to a decision tree is correct when for every possible input, the source program and its respective decision tree produces the same result

$$\forall v_S, t_S(v_S) = \llbracket t_S \rrbracket_S(v_S)$$

Likewise, for the target language:

$$\forall v_T, t_T(v_T) = \llbracket t_T \rrbracket_T(v_T)$$

Definition: in the presence of guards we can say that two results are equivalent modulo the guards queue, written $r_1 \simeq_{gs} r_2$, when:

$$(gs_1, r_1) \simeq_{gs} (gs_2, r_2) \Leftrightarrow (gs_1, r_1) = (gs_2 ++ gs, r_2)$$

Definition: we say that C_T covers the input space S , written $\text{covers}(C_T, S)$ when every value $v_S \in S$ is a valid input to the decision tree C_T . (TODO: rephrase)

Theorem: Given an input space S and a couple of decision trees, where the target decision tree C_T covers the input space S , we say that the two decision trees are equivalent when:

$$\text{equiv}(S, C_S, C_T, gs) = \text{Yes} \wedge \text{covers}(C_T, S) \rightarrow \forall v_S \simeq_{v_T} \in S, C_S(v_S) \simeq_{gs} C_T(v_T)$$

Similarly we say that a couple of decision trees in the presence of an input space S are *not* equivalent when:

$$\text{equiv}(S, C_S, C_T, gs) = \text{No}(v_S, v_T) \wedge \text{covers}(C_T, S) \rightarrow v_S \simeq_{v_T} \in S \wedge C_S(v_S) \not\simeq_{gs} C_T(v_T)$$

Corollary: For a full input space S , that is the universe of the target program we say:

$$\text{equiv}(S, \llbracket t_S \rrbracket_S, \llbracket t_T \rrbracket_T, \emptyset) = \text{Yes} \Leftrightarrow t_S \simeq t_T$$

1. Proof of the correctness of the translation from source programs to source decision trees

We define a source term t_S as a collection of patterns pointing to blackboxes

$$t_S ::= (p \rightarrow \text{bb})^{i \in I}$$

A pattern is defined as either a constructor pattern, an or pattern or a constant pattern

$$p ::= K(p_i)^i, i \in I \quad (p \quad q) \quad n \in \mathbb{N}$$

A decision tree is defined as either a Leaf, a Failure terminal or an intermediate node with different children sharing the same accessor a and an optional fallback. Failure is emitted only when the patterns don't cover the whole set of possible input values S . The fallback is not needed when the user doesn't use a wildcard pattern. %%% Give example of thing

$$\begin{aligned} C_S &::= \text{Leaf } \text{bb} \quad \text{Node}(a, (K_i \rightarrow C_i)^{i \in S}, C?) \\ a &::= \text{Here} \quad \text{n.a} \\ v_S &::= K(v_i)^{i \in I} \quad n \in \mathbb{N} \end{aligned}$$

We define the decomposition matrix m_S as

$$\text{SMatrix } m_S := (a_j)^{j \in J}, ((p_{ij})^{j \in J} \rightarrow \text{bb}_i)^{i \in I}$$

We define the decision tree of source programs $\llbracket t_S \rrbracket$ in terms of the decision tree of pattern matrices $\llbracket m_S \rrbracket$ by the following: $\llbracket ((p_i \rightarrow \text{bb}_i)^{i \in I}) \rrbracket := \llbracket (\text{Root}), (p_i \rightarrow \text{bb}_i)^{i \in I} \rrbracket$

decision tree computed from pattern matrices respect the following invariant:

$$\forall v (v_i)^{i \in I} = v(a_i)^{i \in I} \rightarrow \llbracket m \rrbracket(v) = m(v_i)^{i \in I} \text{ for } m = ((a_i)^{i \in I}, (r_i)^{i \in I})$$

where

$$\begin{aligned} v(\text{Here}) &= v \\ K(v_i)^i(k.a) &= v_k(a) \text{ if } k \in [0;n[\end{aligned}$$

We proceed to show the correctness of the invariant by a case analysis.

Base cases:

- (a) $[[\emptyset, (\emptyset \rightarrow \text{bb}_i)^i]]$:= Leaf bb_i where $i := \min(I)$, that is a decision tree $[[\text{ms}]]$ defined by an empty accessor and empty patterns pointing to blackboxes bb_i . This respects the invariant because a decomposition matrix in the case of empty rows returns the first expression and we know that $(\text{Leaf } \text{bb})(v) := \text{Match } \text{bb}$
- (b) $[[(a_j)^j, \emptyset]]$:= Failure

Regarding non base cases: Let's first define

$$\begin{aligned} \text{let } \text{Idx}(k) &:= [0; \text{arity}(k)[\\ \text{let } \text{First}(\emptyset) &:= \perp \\ \text{let } \text{First}((a_j)^j) &:= a_{\min(j \in J \neq \emptyset)} \end{aligned}$$

$$m := ((a_i)^i((p_{ij})^i \rightarrow e_j)^{ij})$$

$$(k_k)^k := \text{headconstructor}(p_{i0})^i$$

$$\text{Groups}(m) := (k_k \rightarrow ((a)_{0..l})^{l \in \text{Idx}(k_k)} + + + (a_i)^{i \in I \setminus \{0\}}, (\text{if } p_{0j} \text{ is } k(q_l) \text{ then } (q_l)^{l \in \text{Idx}(k_k)} + + + (p_{ij})^{i \in I \setminus \{0\}})$$

(1)

$\text{Groups}(m)$ is an auxiliary function that decomposes a matrix m into submatrices, according to the head constructor of their first pattern. $\text{Groups}(m)$ returns one submatrix m_r for each head constructor k that occurs on the first row of m , plus one "wildcard submatrix" m_{wild} that matches on all values that do not start with one of those head constructors.

Intuitively, m is equivalent to its decomposition in the following sense: if the first pattern of an input vector $(v_i)^i$ starts with one of the head

constructors k , then running $(v_i)^l$ against m is the same as running it against the submatrix m_k ; otherwise (its head constructor is none of the k) it is equivalent to running it against the wildcard submatrix.

We formalize this intuition as follows: Lemma (Groups): Let

$$m$$

be a matrix with

$$Groups(m) = (k_r \rightarrow m_r)^k, m_{wild}$$

. For any value vector

$$(v_i)^l$$

such that

$$v_0 = k(v_i)^l$$

for some constructor k , we have:

$$if k = k_k for some k then m(v_i)^i = m_k((v_i)^l + \dots + (v_i)^{i \in I \setminus \{0\}}) else m(v_i)^i = m_{wild}(v_i)^{i \in I \setminus \{0\}}$$

2. Proof: Let

$$m$$

be a matrix with

$$Group(m) = (k_r \rightarrow m_r)^k, m_{wild}$$

. Let

$$(v_i)^i$$

be an input matrix with

$$v_0 = k(v_i)^i$$

for some k . We proceed by case analysis:

- either k is one of the k_k for some k
- or k is none of the $(k_k)^k$

Both $m(v_i)^i$ and $m_k(v_k)^k$ are defined as the first matching result of a family over each row r_j of a matrix

We know, from the definition of Groups(m), that m_k is

$$((a)0.l)^{l \in \text{Idx}(k_k)} \text{ } \text{ } \text{ } (a_i)^{i \in I \setminus \{0\}}, (if p_0 j isk(q_l) then (q_l)^l \text{ } \text{ } \text{ } (p_{ij})^{i \in I \setminus \{0\}} \rightarrow e_j if p_0 j isthen^l \text{ } \text{ } \text{ } (p_{ij})^{i \in I \setminus \{0\}}$$

By definition, $m(v_i)^i$ is $m(v_i)^i = \text{First}(r_j(v_i)^i)^j$ for $m = ((a_i)^i, (r_j)^j)$
 $(p_i)^i (v_i)^i = \{$ if $p_0 = k(q_l)^l, v_0 = k'(v'_k)^k, k = \text{Idx}(k')$ and $l = \text{Idx}(k)$ if
 $k \neq k'$ then \perp if $k = k'$ then $((q_l)^l + (p_i)^{i \in I \setminus \{0\}}) ((v'_k)^k + (v_i)^{i \in I \setminus \{0\}})$ if
 $p_0 = (q_1 | q_2)$ then $\text{First}((q_1 p_i)^{i \in I \setminus \{0\}}, (q_2 p_i)^{i \in I \setminus \{0\}})$
 $\left. \right\}$

For this reason, if we can prove that

$$\forall j, r_j(v_i)^i = r'_j((v'_k)^k \text{ } \text{ } (v_i)^i)$$

it follows that

$$m(v_i)^i = m_k((v'_k)^k \text{ } \text{ } (v_i)^i)$$

from the above definition.

We can also show that $a_i = a_{0.l}^l + a_{i \in I \setminus \{0\}}$ because $v(a_0) = K(v(a)\{0.l\})^l$

1.2 Proof of equivalence checking

1.2.1 The trimming lemma

The trimming lemma allows to reduce the size of a decision tree given an accessor $\rightarrow \pi$ relation (TODO: expand)

$$\forall v_T \in (a \rightarrow \pi), C_T(v_T) = C_{t/a \rightarrow \pi(k_i)}(v_T)$$

We prove this by induction on C_T : a. $C_T = \text{Leaf}_{bb}$: when the decision tree is a leaf terminal, we know that

$$\text{Leaf}_{bb/a \rightarrow \pi}(v) = \text{Leaf}_{bb}(v)$$

That means that the result of trimming on a Leaf is the Leaf itself b . The same applies to Failure terminal

$$\text{Failure}_{/a \rightarrow \pi}(v) = \text{Failure}(v)$$

c. When $C_T = \text{Node}(b, (\pi \rightarrow C_i)^i)_{/a \rightarrow \pi}$ then we look at the accessor a of the subtree C_i and we define $\pi_i' = \pi_i$ if $a \neq b$ else $\pi_i \cap \pi$. Trimming a switch node yields the following result:

$$\text{Node}(b, (\pi \rightarrow C_i)^i)_{/a \rightarrow \pi} := \text{Node}(b, (\pi_i' \rightarrow C_{i/a \rightarrow \pi})^i)$$

For the trimming lemma we have to prove that running the value v_T against the decision tree C_T is the same as running v_T against the tree C_{trim} that is the result of the trimming operation on C_T

$$C_T(v_T) = C_{\text{trim}}(v_T) = \text{Node}(b, (\pi_i' \rightarrow C_{i/a \rightarrow \pi})^i)(v_T)$$

We can reason by first noting that when $v_T \notin (b \rightarrow \pi_i)^i$ the node must be a Failure node. In the case where $\exists k \mid v_T \in (b \rightarrow \pi_k)$ then we can prove that

$$C_{k/a \rightarrow \pi}(v_T) = \text{Node}(b, (\pi_i' \rightarrow C_{i/a \rightarrow \pi})^i)(v_T)$$

because when $a \neq b$ then $\pi_k' = \pi_k$ and this means that $v_T \in \pi_k'$ while when $a = b$ then $\pi_k' = (\pi_k \cap \pi)$ and $v_T \in \pi_k'$ because:

- by the hypothesis, $v_T \in \pi$
- we are in the case where $v_T \in \pi_k$

So $v_T \in \pi_k'$ and by induction

$$C_k(v_T) = C_{k/a \rightarrow \pi}(v_T)$$

We also know that $\forall v_T \in (b \rightarrow \pi_k) \rightarrow C_T(v_T) = C_k(v_T)$. By putting together the last two steps, we have proven the trimming lemma.

1.2.2 Equivalence checking

The equivalence checking algorithm takes as parameters an input space S , a source decision tree C_S and a target decision tree C_T :

$$\text{equiv}(S, C_S, C_T) \rightarrow \text{Yes} \quad \text{No}(v_S, v_T)$$

When the algorithm returns Yes and the input space is covered by C_S we can say that the couple of decision trees are the same for every couple of source value v_S and target value v_T that are equivalent.

$$\text{equiv}(S, C_S, C_T) = \text{Yes and cover}(C_T, S) \rightarrow \forall v_S \simeq v_T \in S \wedge C_S(v_S) = C_T(v_T)$$

In the case where the algorithm returns No we have at least a couple of counter example values v_S and v_T for which the two decision trees outputs a different result.

$$\text{equiv}(S, C_S, C_T) = \text{No}(v_S, v_T) \text{ and cover}(C_T, S) \rightarrow \forall v_S \simeq v_T \in S \wedge C_S(v_S) \neq C_T(v_T)$$

We define the following

$$\begin{aligned} \text{Forall}(\text{Yes}) &= \text{Yes} \\ \text{Forall}(\text{Yes}::l) &= \text{Forall}(l) \\ \text{Forall}(\text{No}(v_S, v_T)::_) &= \text{No}(v_S, v_T) \end{aligned}$$

There exists and are injective:

$$\begin{aligned} \text{int}(k) &\in \mathbb{N} \quad (\text{arity}(k) = 0) \\ \text{tag}(k) &\in \mathbb{N} \quad (\text{arity}(k) > 0) \\ \pi(k) &= \{n \mid \text{int}(k) = n\} \times \{n \mid \text{tag}(k) = n\} \end{aligned}$$

where k is a constructor.

We proceed by case analysis:

1. in case of unreachable:

$$C_S(v_S) = \text{Absurd}(\text{Unreachable}) \neq C_T(v_T) \quad \forall v_S, v_T$$

1. In the case of an empty input space

$$\text{equiv}(\emptyset, C_S, C_T) := \text{Yes}$$

and that is trivial to prove because there is no pair of values (v_S, v_T) that could be tested against the decision trees. In the other subcases S is always non-empty.

2. When there are *Failure* nodes at both sides the result is *Yes*:

$$\text{equiv}(S, \text{Failure}, \text{Failure}) := \text{Yes}$$

Given that $\forall v, \text{Failure}(v) = \text{Failure}$, the statement holds.

3. When we have a Leaf or a Failure at the left side:

$$\begin{aligned} \text{equiv}(S, \text{Failure as } C_S, \text{Node}(a, (\pi_i \rightarrow C_{T_i})^i)) &:= \text{Forall}(\text{equiv}(S \cap a \rightarrow \pi(k_i)), C_S, C_{T_i})^i \\ \text{equiv}(S, \text{Leaf } \text{bb}_S \text{ as } C_S, \text{Node}(a, (\pi_i \rightarrow C_{T_i})^i)) &:= \text{Forall}(\text{equiv}(S \cap a \rightarrow \pi(k_i)), C_S, C_{T_i})^i \end{aligned}$$

The algorithm either returns *Yes* for every sub-input space $S_i := S \cap (a \rightarrow \pi(k_i))$ and subtree C_{T_i}

$$\text{equiv}(S_i, C_S, C_{T_i}) = \text{Yes } \forall i$$

or we have a counter example v_S, v_T for which

$$v_S \simeq v_T \in S_k \wedge c_S(v_S) \neq C_{T_k}(v_T)$$

then because

$$\begin{aligned} v_T \in (a \rightarrow \pi_k) \rightarrow C_T(v_T) &= C_{T_k}(v_T) , \\ v_S \simeq v_T \in S \wedge C_S(v_S) &\neq C_T(v_T) \end{aligned}$$

we can say that

$$\text{equiv}(S_i, C_S, C_{T_i}) = \text{No}(v_S, v_T) \text{ for some minimal } k \in I$$

4. When we have a Node on the right we define π_n as the domain of values not covered but the union of the constructors k_i

$$\pi_n = \neg(\bigcup \pi(k_i)^i)$$

The algorithm proceeds by trimming

$$\begin{aligned} & \text{equiv}(S, \text{Node}(a, (k_i \rightarrow C_{S_i})^i, C_{sf}), C_T) := \\ & \text{Forall}(\text{equiv}(S \cap (a \rightarrow \pi(k_i)^i), C_{S_i}, C_{t/a \rightarrow \pi(k_i)^i})^i + \text{equiv}(S \cap (a \rightarrow \pi(k_i)), C_S, C_{a \rightarrow \pi_n})) \end{aligned}$$

The statement still holds and we show this by first analyzing the *Yes* case:

$$\text{Forall}(\text{equiv}(S \cap (a \rightarrow \pi(k_i)^i), C_{S_i}, C_{t/a \rightarrow \pi(k_i)^i})^i = \text{Yes}$$

The constructor k is either included in the set of constructors k_j :

$$k \mid k \in (k_i)^i \wedge C_S(v_S) = C_{S_i}(v_S)$$

We also know that

$$\begin{aligned} (1) \quad & C_{S_i}(v_S) = C_{t/a \rightarrow \pi_i}(v_T) \\ (2) \quad & C_{T/a \rightarrow \pi_i}(v_T) = C_T(v_T) \end{aligned}$$

(1) is true by induction and (2) is a consequence of the trimming lemma. Putting everything together:

$$C_S(v_S) = C_{S_i}(v_S) = C_{T/a \rightarrow \pi_i}(v_T) = C_T(v_T)$$

When the $k \notin (k_i)^i$ [TODO]

The auxiliary Forall function returns $No(v_S, v_T)$ when, for a minimum k ,

$$\text{equiv}(S_k, C_{S_k}, C_{T/a \rightarrow \pi_k}) = No(v_S, v_T)$$

Then we can say that

$$C_{Sk}(v_S) \neq C_{t/a \rightarrow \pi_k}(v_T)$$

that is enough for proving that

$$C_{Sk}(v_S) \neq (C_{t/a \rightarrow \pi_k}(v_T) = C_T(v_T))$$