Francesco Mecca

1 Correctness of the algorithm

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

$$([\![t_S]\!]_S(v_S) = C_S(v_S)) \to r$$
$$t_S(v_S) \to r$$

Likewise

$$([\![t_T]\!]_T(v_T) = C_T(v_T)) \to r$$
$$t_T(v_T) \to r$$

where result r ::= guard list * (Match blackbox | NoMatch | Absurd) and guard ::= 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

$$t_S \simeq t_T := \forall v_S \simeq v_T,\ t_S(v_S) = t_T(v_T)$$
$$C_S \simeq C_T := \forall v_S \simeq v_T,\ C_S(v_S) = C_T(v_T)$$

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

$$\forall v_S,\ t_S(v_S) = [\![t_S]\!]_S(v_S)$$

Likewise, for the target language:

$$\forall v_T,\ t_T(v_T) = [\![t_T]\!]_T(v_T)$$

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:

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

Definition: we say that C_T covers the input space S, written /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:

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

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

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

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

$$\text{equiv}(S,\ [\![t_S]\!]_S,\ [\![t_T]\!]_T,\ \varnothing) = \text{Yes} \Leftrightarrow t_S \simeq t_T$$

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

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

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

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

   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

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

   We define the decomposition matrix m_S as

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

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

   decision tree computed from pattern matrices respect the following invariant:

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

where

$$\mathrm{v(Here)} = \mathrm{v}$$
$$\mathrm{K}(\mathrm{v}_i)^i(\mathrm{k.a}) = \mathrm{v}_k(\mathrm{a}) \text{ if } \mathrm{k} \in [0;\mathrm{n}[$$

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

Base cases:

(a) $[| \varnothing, (\varnothing \rightarrow \mathrm{bb}_i)^i |] := \mathrm{Leaf}\ \mathrm{bb}_i$ where i := min(I), that is a decision tree [|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 known that (Leaf bb)(v) := Match bb

(b) $[| (\mathrm{a}_j)^j, \varnothing |] := \mathrm{Failure}$

Regarding non base cases: Let's first define

$$\text{let Idx(k)} := [0; \text{arity(k)}[$$
$$\text{let First}(\varnothing) := \bot$$
$$\text{let First}((\mathrm{a}_j)^j) := \mathrm{a}_{\min(\mathrm{j} \in \mathrm{J} \neq \varnothing)}$$

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

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

$$Groups(m) := (k_k \rightarrow ((a)_{0.l})^{l \in Idx(k_k)} {+}{+}{+} (a_i)^{i \in I \setminus \{0\}}), (if p_{0j} is k(q_l) then (q_l)^{l \in Idx(k_k)} {+}{+}{+} (p_{ij})^{i \in I \setminus \{ \tag{1}$$

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

Intuitively, m is equivalent to its decompositionin 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)^i 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

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

. For any value vector

$$(v_i)^l$$

such that

$$v_0 = k(v'_l)^l$$

for some constructor k, we have:

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

2. Proof: Let

$$m$$

be a matrix with

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

. Let

$$(v_i)^i$$

be an input matrix with

$$v_0 = k(v'_l)^l$$

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

$$((a)0.l)^{l \in Idx(k_k)} {+}{+}{+} (a_i)^{i \in I \setminus \{0\}}), (if p_{0j} is k(q_l) then (q_l)^l {+}{+}{+} (p_{ij})^{i \in I \setminus \{0\}} \rightarrow e_j if p_{0j} is_t hen (^l_) {+}{+}{+} (p_$$

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

For this reason, if we can prove that

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

it follows that

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

from the above definition.

We can also show that $a_i = a_{0.1}{}^l + a_{i \in I \setminus \backslash 0 \backslash}$ because $v(a_0) = K(v(a)\{0.1\})^l)$

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. $C_T = \mathrm{Leaf}_{bb}$: when the decision tree is a leaf terminal, we know that

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

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

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

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

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

For the trimming lemma we have to prove that running the value v_T against the decistion 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

$$\text{C}_T(\text{v}_T) = \text{C}_{\text{trim}}(\text{v}_T) = \text{Node(b, }(\pi_i\text{'}\rightarrow\text{C}_{\text{i}/\text{a}\rightarrow\pi})^i)(\text{v}_T)$$

We can reason by first noting that when $\text{v}_T\notin(\text{b}\rightarrow\pi_i)^i$ the node must be a Failure node. In the case where ∃k| $\text{v}_T\in(\text{b}\rightarrow\pi_k)$ then we can prove that

$$\text{C}_{\text{k}/\text{a}\rightarrow\pi}(\text{v}_T) = \text{Node(b, }(\pi_i\text{'}\rightarrow\text{C}_{\text{i}/\text{a}\rightarrow\pi})^i)(\text{v}_T)$$

because when a ≠ b then π_k'= π_k and this means that $\text{v}_T\in\pi_k$' while when a = b then π_k'=$(\pi_k\cap\pi)$ and vt$\in\pi_k$' because:

- by the hypothesis, $\text{v}_T\in\pi$
- we are in the case where $\text{v}_T\in\pi_k$

So $\text{v}_T \in \pi_k$' and by induction

$$\text{C}_k(\text{v}_T) = \text{C}_{\text{k}/\text{a}\rightarrow\pi}(\text{v}_T)$$

We also know that $\forall\text{v}_T\in(\text{b}\rightarrow\pi_k) \rightarrow \text{C}_T(\text{v}_T) = \text{C}_k(\text{v}_T)$ 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:

$$\text{equiv(S, C}_S\text{, C}_T\text{)} \rightarrow \text{Yes} \quad \text{No(v}_S\text{, v}_T\text{)}$$

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.

$$\text{equiv(S, C}_S\text{, C}_T\text{) = Yes and cover(C}_T\text{, S)} \rightarrow \forall \text{ v}_S \simeq \text{v}_T \in \text{S} \land \text{C}_S\text{(v}_S\text{) = C}_T\text{(v}_T\text{)}$$

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.

$$\text{equiv(S, C}_S\text{, C}_T\text{) = No(v}_S\text{,v}_T\text{) and cover(C}_T\text{, S)} \rightarrow \forall \text{ v}_S \simeq \text{v}_T \in \text{S} \land \text{C}_S\text{(v}_S\text{)} \neq \text{C}_T\text{(v}_T\text{)}$$

We define the following

$$\begin{aligned}
&\text{Forall(Yes) = Yes} \\
&\text{Forall(Yes::l) = Forall(l)} \\
&\text{Forall(No(v}_S\text{,v}_T\text{)::_) = No(v}_S\text{,v}_T\text{)}
\end{aligned}$$

There exists and are injective:

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

where k is a constructor.

We proceed by case analysis:

1. in case of unreachable:

$$\text{C}_S\text{(v}_S\text{) = Absurd(Unreachable)} \neq \text{C}_T\text{(v}_T\text{)} \ \forall \text{v}_S\text{,v}_T$$

1. In the case of an empty input space

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

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 ∀v, Failure(v) = Failure, the statement holds.

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

$$\text{equiv}(S,\ \text{Failure as } C_S,\ \text{Node}(a,\ (\pi_i \to C_{Ti})^i)) := \text{Forall}(\text{equiv}(\ S{\cap}a{\to}\pi(k_i)),\ C_S,\ C_{Ti})^i)$$
$$\text{equiv}(S,\ \text{Leaf bb}_S \text{ as } C_S,\ \text{Node}(a,\ (\pi_i \to C_{Ti})^i)) := \text{Forall}(\text{equiv}(\ S{\cap}a{\to}\pi(k_i)),\ C_S,\ C_{Ti})^i)$$

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

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

or we have a counter example v_S, v_T for which

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

then because

$$v_T{\in}(a{\to}\pi_k) \to C_T(v_T) = C_{Tk}(v_T)\ ,$$
$$v_S{\simeq}v_T{\in}S \wedge C_S(v_S){\neq}C_T(v_T)$$

we can say that

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

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

$$\text{equiv}(S, \text{Node}(a, (k_i \to C_{Si})^i, C_{sf}), C_T) :=$$
$$\text{Forall}(\text{equiv}(S\cap(a\to\pi(k_i)^i), C_{Si}, C_{t/a\to\pi(k_i)})^i + \text{equiv}(S\cap(a\to\pi(k_i)), C_S, C_{a\to\pi_n}))$$

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

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

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

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

We also know that

$$(1)\ C_{Si}(v_S) = C_{t/a\to\pi_i}(v_T)$$
$$(2)\ C_{T/a\to\pi_i}(v_T) = C_T(v_T)$$

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

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

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

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

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

Then we can say that

$$\mathrm{C}_{Sk}(\mathrm{v}_S) \neq \mathrm{C}_{\mathrm{t/a}\to\pi_k}(\mathrm{v}_T)$$

that is enough for proving that

$$\mathrm{C}_{Sk}(\mathrm{v}_S) \neq (\mathrm{C}_{\mathrm{t/a}\to\pi_k}(\mathrm{v}_T) = \mathrm{C}_T(\mathrm{v}_T))$$