

ELEMENTS OF PROGRAMMING SOLUTIONS

ASEF AHMED

1. FOUNDATIONS

1.1. Categories of Ideas: Entity, Species, Genus.

1.2. Values.

Lemma 1.1. *If a value type is uniquely represented, equality implies representational equality.*

Solution. Suppose a value type T is uniquely represented. Denote by $v, v' : T$ as equal values of T . By unique representation, v, v' each correspond uniquely to the abstract entities E, E' , and by equality of values, these entities must also be equal. Hence the data D, D' for v, v' are identical, and so v, v' are representationally equal. \square

Lemma 1.2. *If a value type is not ambiguous, representational equality implies equality.*

Solution. Suppose a value type T is not ambiguous. Denote by $v, v' : T$ as representationally equal values of T . As T is not ambiguous, v, v' must each have at most one interpretation, and by representational equality, the data D, D' for the values are identical. Hence the values v, v' must represent the same abstract entity E , and so they are equal. \square

1.3. Objects.

1.4. Procedures.

1.5. Regular Types.

Lemma 1.3. *A well-formed object is partially formed.*

Solution. Suppose a is an object that is well-formed. Let $S(a)$ be the state of a , which by definition is a value $v : T$ of some value type T . By well-formedness of a , $S(a)$ is also well-formed as a value, i.e. WLOG, we may reduce to the case where T is `double` as an object type for a . Let b be another object of type `double`. Certainly a may be assigned to b without modifying the state $S(b)$ of b , and a may be destroyed as well. Therefore a is partially formed. \square

1.6. Regular Procedures.

Exercise 1.1. *Extend the notion of regularity to input/output objects of a procedure, that is, to objects that are modified as well as read.*

Solution. A procedure is *regular* if and only if the input objects a_0, \dots, a_r , when replaced by equal objects b_0, \dots, b_r , i.e. for each $i \in \{0, \dots, r\}$, the states $S(a_i) = S(b_i)$ are equivalent for the objects a_i, b_i of type T_i , yield equivalent output objects $c_i = d_i$ (where equality of objects means equivalence of the corresponding states). There is a natural equivalence for input/output objects in the following way: an input/output object s is equivalent to t , i.e. $s \equiv t$, if there is a regular procedure F for which t is the output object of the input s under F , or symmetrically if s is the output object of the input t under F . \square

Exercise 1.2. *Assuming that `int` is a 32-bit two's complement type, determine the exact definition and result space.*

Solution. Supposing `int` is 32-bit two's complement, i.e. `int` is signed, the exact definition space would consist of the closed interval $I := [-46340, 46340]$ and the result space would consist of the set $\{x * x | x : \text{int}, x \in I\}$, i.e. $\{1, 4, 9, \dots, 46340 * 46340\}$. \square

2. TRANSFORMATIONS AND THEIR ORBITS

2.1. Transformations.

Lemma 2.1. `euclidean_norm(x, y, z) = euclidean_norm(euclidean_norm(x, y), z)`

Solution. `euclidean_norm(x, y, z)` is a ternary operation with the signature

`euclidean_norm :: double × double × double → double`

With currying, the signature becomes

`euclidean_norm :: double → double → double → double`

which clearly associates to

`euclidean_norm :: (double → double → double) → double`

Now, one can un-curry the parenthesized to obtain

`euclidean_norm :: (double × double → double) → double`

Where, now, the parenthesized is simply the binary operation `euclidean_norm(x, y)`, which in this case can be taken as a partial application of the function `euclidean_norm(x, y, z)` to the arguments `x, y`. So it is possible to "build" a ternary operation from a binary operation. \square