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

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

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

1.3. Objects.

1.4. Procedures.

1.5. Regular Types.

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

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

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 $\mathtt{a\_0}, \ldots, \mathtt{a\_r}$, when replaced by equal objects $\mathtt{b\_0}, \ldots, \mathtt{b\_r}$, i.e. for each $i \in \{0, \ldots, r\}$, the states $\mathtt{S}(\mathtt{a\_i}) = \mathtt{S}(\mathtt{b\_i})$ are equivalent for the objects $\mathtt{a\_i}, \mathtt{b\_i}$ of type $\mathtt{T\_i}$, yield equivalent output objects $\mathtt{c\_i} = \mathtt{d\_i}$ (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 $\mathtt{s}$ is equivalent to $\mathtt{t}$, i.e. $\mathtt{s} \equiv \mathtt{t}$, if there is a regular procedure $\mathtt{F}$ for which $\mathtt{t}$ is the output object of the input $\mathtt{s}$ under $\mathtt{F}$, or symmetrically if $\mathtt{s}$ is the output object of the input $\mathtt{t}$ under $\mathtt{F}$. □

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 $\mathtt{I} := [-\mathtt{46340}, \mathtt{46340}]$ and the result space would consist of the set $\{\mathtt{x} * \mathtt{x} | \mathtt{x} : \mathtt{int}, \mathtt{x} \in \mathtt{I}\}$, i.e. $\{\mathtt{1}, \mathtt{4}, \mathtt{9}, \ldots, \mathtt{46340} * \mathtt{46340}\}$. □

2. Transformations and Their Orbits

2.1. Transformations.

Lemma 2.1. $\mathtt{euclidean\_norm}(\mathtt{x}, \mathtt{y}, \mathtt{z}) = \mathtt{euclidean\_norm}(\mathtt{euclidean\_norm}(\mathtt{x}, \mathtt{y}), \mathtt{z})$

Solution. $\mathtt{euclidean\_norm}(\mathtt{x}, \mathtt{y}, \mathtt{z})$ is a ternary operation with the signature

$$\mathtt{euclidean\_norm} :: \mathtt{T} \times \mathtt{T} \times \mathtt{T} \to \mathtt{T}$$

where `T` is an *arithmetic type* (viz. a type on which arithmetic operations such as $+, *, \sqrt{\cdot}$ can be performed, e.g. unary and binary functions whose domains and codomains are integral or floating-point types, though not always operations: $\sqrt{\cdot}$ could possibly map an element from a domain of integral type to an element in a codomain of floating-point type). After currying, the signature becomes

$$\mathtt{euclidean\_norm} :: \mathtt{T} \to \mathtt{T} \to \mathtt{T} \to \mathtt{T}$$

which clearly associates to

$$\mathtt{euclidean\_norm} :: \mathtt{T} \to (\mathtt{T} \to \mathtt{T} \to \mathtt{T})$$

Now, one can un-curry the parenthesized to obtain

$$\mathtt{euclidean\_norm} :: \mathtt{T} \to (\mathtt{T} \times \mathtt{T} \to \mathtt{T})$$

Where, now, the parenthesized is simply the binary operation $\mathtt{euclidean\_norm}(\mathtt{x}, \mathtt{y})$, which may be taken as a partial application of the function $\mathtt{euclidean\_norm}(\mathtt{x}, \mathtt{y}, \mathtt{z})$ to the arguments $\mathtt{x}, \mathtt{y}$, upon which one can then apply the third argument `z` to obtain the Euclidean norm of all three. So it is possible to "build" a ternary operation from a binary operation. □