Neutral comparators

Jakob Wuhrer

2022-11-23

In this document, we will examine the possibility of defining neutral elements for srfi 228's "product" and "sum" operators on srfi 128 style comparators.

1 Notation and conventions

In this document, we will refer to the set of all scheme values/objects (is there a difference?) as U.

Furthermore, we will represent srfi 128 style comparators as tuples of

- a characteristic function for a subset of U,
- a characteristic function for an equivalence relation on the corresponding set,
- a characteristic function for $<$ where $<$ is a partial order on the corresponding set, and
- a hashing function that maps elements of the aforementioned corresponding set into $\mathbb{N}$.

Note that a tuple of such functions is a comparator if and only if the order induces a total order on the quotient set w.r.t. the equivalence relation, and the hashing function induces a hashing function on the quotient set w.r.t. the equivalence relation.

In keeping with common scheme style, we will write $\#t$ and $\#f$ to refer to the two scheme values representing truth and falsehood. These will, in some cases, take on the role that 0 and 1 usually play in mathematical texts.

If $A \subseteq B$ is a subset, then we may write χ_A to refer to the characteristic function of A, that is, the following function:

$$\chi_A : B \to B, b \mapsto \begin{cases} \#t & b \in A \\ \#f & b \notin A \end{cases}.$$

For any scheme value $u \in U$ and for any integer $n \in$, we may refer to the function $U^n \to U, \vec{x} \mapsto u$ as $\text{const}_n(u)$.

If $\mathcal{A}, \mathcal{B}$ are two comparators in the sense of srfi 128, then we may write $\mathcal{A} \otimes \mathcal{B}$ to refer to the srfi 228 product of the two.

Similarly, if $\mathcal{A}, \mathcal{B}$ are two comparators in the sense of srfi 128, then we may write $\mathcal{A} \oplus \mathcal{B}$ to refer to the srfi 228 sum of the two.

Note that these operations are not necessarily commutative.

2 One

We will now shift our attention to the product operation. Our aim is to define a comparator that is both a left and a right neutral element for $\otimes$.

Recall that U refers to the entire set of all scheme values, and note that $\text{const}_1(\#t) = \chi_U$.

Furthermore, note that $\text{const}_2(\#t)$ forms a (trivial) equivalence relation on U, its quotient is the singleton with as its sole element the class containing all elements of U. We may write $\sim_U$ to refer to $\text{const}_2(\#t)$.

Once we have this quotient, note that the the trivial partial order on U (where no element is greater than any other element) induces a (trivial) total order on the aforementioned quotient set. The function $\text{const}_2(\#f)$ is the $<$-relation for this order. We may write $<_U$ to refer to $\text{const}_2(\#f)$.

Finally, note that the function $\text{const}_1(0)$ has the required "hashing" property, that is, for any $a, b \in U$ with $\text{const}_2(\#t)(a, b)$, we have $\text{const}_1(0)(a) = \text{const}_1(0)(b)$. We may write $\#_U$ to refer to $\text{const}_2(\#t)$

Thus, the following tuple is a comparator in the sense of srfi 128:

$$\begin{aligned} \mathcal{U} &= (\chi_U, \sim_U, <_U, \#_U) \\ &= (\text{const}_1(\#t), \text{const}_2(\#t), \text{const}_2(\#f), \text{const}_1(0)). \end{aligned}$$

Let a set $A \subseteq U$ of scheme values be given, and let $\sim$ be an equivalence relation on A. Furthermore, let a partial order $<$ be given such that the

induced order on ${}^{A}/_{\sim}$ is total. Finally, let a hashing function $\# : A \to \mathbb{N}$ be given, such that $\forall a, b \in A$ we have $a \sim b \implies \#(a) = \#(b)$.

Then, the tuple $\mathcal{A} = (\chi_A, \sim, <, \#)$ is a comparator in the sense of srfi 128.

2.1 Left

Now, let us show that $\mathcal{U} \otimes \mathcal{A} = \mathcal{A}$.

Let us first write $\mathcal{B} = \mathcal{U} \otimes \mathcal{A}$, let B be the set corresponding to the first element of the tuple $\mathcal{B}$, let $\sim_B$ be the equivalence relation corresponding to its second element, let $<_B$ be the partial order corresponding to its third element, and let $\#_B$ be the hashing function corresponding to its final element.

2.1.1 Characteristic / type test

Note that a product comparator is defined on the intersection of the sets where its factor comparators are defined. Therefore, $\mathcal{B}$ is defined exactly where $\mathcal{A}$ is defined. Let us state this more formally:

Let $x \in U$ be arbitrarily given.

Then, by the definition of `make-product-comparator`, we have that:

$$\begin{aligned}\chi_B(x) &= \chi_A(x) \wedge \chi_U(x) \\ &= \chi_A(x) \wedge \mathrm{const}_1(\#t)(x) \\ &= \chi_A(x) \wedge \#t \\ &= \chi_A(x).\end{aligned}$$

2.1.2 Equivalence relation

Let $x, y \in (A \cap U) = A$ be arbitrarily given. Then, as described above, both x and y are of $\mathcal{B}$'s type.

Furthermore, by the definition of `make-product-comparator`, we have that:

$$\begin{aligned}x \sim_B y &= (x \sim_A y) \wedge \mathrm{const}_2(\#t)(x, y) \\ &= (x \sim_A y) \wedge \#t \\ &= x \sim_A y\end{aligned}$$

2.1.3 Order

Let $x, y \in (A \cap U) = A$ be arbitrarily given. Then, as described above, both x and y are of $\mathcal{B}$'s type.

Furthermore, by the definition of `make-product-comparator`, we have that:

$$\begin{aligned} x <_B y &= (!(x \sim_U y) \wedge (x \leq_U y)) \vee ((x \sim_U y) \wedge (x \leq_A y)) \\ &= (!\text{const}_2(\#t)(x, y) \wedge \text{const}_2(\#f)(x, y)) \vee (\text{const}_2(\#t)(x, y) \wedge (x \leq_A y)) \\ &= (!\#t \wedge \#f) \vee (\#t \wedge (x \leq_A y)) \\ &= x \leq_A y \end{aligned}$$

2.1.4 Hashing

Note that, since the behavior of $\otimes$ with respect to hashing is not very strictly specified, it is not generally true for all conforming definitions of $\otimes$ that $\#_B = \#_A$. However, this is true in the case of the example implementation. I personally don't think that this is much of an issue, since it is possible to adapt my definition of $\#_U$ to any other implementation as long as there is a neutral value for the function used to "combine" two hashes.

Let $x \in (A \cap A) = A$ be arbitrarily given. Then, as described above, x is of $\mathcal{B}$'s type.

Furthermore, by the definition of `make-product-comparator`, we have

$$\begin{aligned} \#_B(x) &= \text{xor}(\#_U(x), \#_A(x)) \\ &= \text{xor}(\text{const}_1(0)(x), \#_A(x)) \\ &= \text{xor}(0, \#_A(x)) \\ &= \#_A(x) \end{aligned}$$

2.1.5 Conclusion

Therefore, for all comparators $\mathcal{A}$, we have $\mathcal{U} \otimes \mathcal{A} = \mathcal{A}$.

2.2 Right

The proof to show that $\mathcal{A} \otimes \mathcal{U} = \mathcal{A}$ is wholly analogous to the left case. Since the functions used to "combine" the type checking, the equivalence checking, and the hashing are commutative, we can simply refer to the proof for the left case. The proof that $<_{\mathcal{A} \otimes \mathcal{U}} = <_A$ is not identical to the proof that $<_{\mathcal{U} \otimes \mathcal{A}} = <_A$, but they closely mirror each other.

2.3 Conclusion

The following comparator is a left and right neutral element for $\otimes$ in the example implementation:

$$\begin{aligned}\mathcal{U} &= (\chi_U, \sim_U, <_U, \#_U) \\ &= (\mathrm{const}_1(\#t), \mathrm{const}_2(\#t), \mathrm{const}_2(\#f), \mathrm{const}_1(0)).\end{aligned}$$

Furthermore, if we ignore hashing, it is a left right neutral element for $\otimes$ in any conforming implementation.

3 Zero

With products done, we will now define a neutral element for $\oplus$. This will prove to be easier.

In the case of the neutral element for $\otimes$, we defined a comparator on the entirety of U. The neutral element for $\oplus$ corresponds to the other trivial subset of U: the empty set.

Note that $\chi_\emptyset = \mathrm{const}_1(\#f)$.

Furthermore, note that there is just one equivalence relation on the empty set, let $\sim_\emptyset$ be this equivalence relation.

Similarly, there is just one partial order on the empty set, let $<_\emptyset$ be this partial order.

Finally, there is exactly one function $\emptyset \to \mathbb{N}$, let $\#_\emptyset$ be this function.

Then, the tuple $\mathcal{Q} = (\chi_\emptyset, \sim_\emptyset, <_\emptyset, \#_\emptyset)$ is comparator in the sense of srfi 128.

Note that we don't mind when $\sim_\emptyset$, $<_\emptyset$, and $\#_\emptyset$ are defined on a larger domain than is required. Therefore, any function of adequate arity is a correct implementation of these functions.

Let, again, a set $A \subseteq U$ of scheme values be given, and let $\sim$ be an equivalence relation on A. Furthermore, let a partial order $<$ be given such that the induced order on ${}^{A}\!/_{\sim}$ is total. Finally, let a hashing function $\# : A \to \mathbb{N}$ be given, such that $\forall a, b \in A$ we have $a \sim b \implies \#(a) = \#(b)$.

Then, the tuple $\mathcal{A} = (\chi_A, \sim, <, \#)$ is a comparator in the sense of srfi 128.

3.1 Left

Now, let us show that $\mathcal{Q} \oplus \mathcal{A} = \mathcal{A}$.

Let us first write $\mathcal{B} = \mathcal{Q} \oplus \mathcal{A}$, let B be the set corresponding to the first element of the tuple $\mathcal{B}$, let $\sim_B$ be the equivalence relation corresponding to its second element, let $<_B$ be the partial order corresponding to its third element, and let $\#_B$ be the hashing function corresponding to its final element.

3.1.1 Characteristic function

Let $x \in U$ be arbitrarily given. By the definition of `make-sum-comparator`, the following is true:

$$\begin{aligned}\chi_B(x) &= \chi_\emptyset(x) \vee \chi_A(x) \\ &= \mathrm{const}_1(\#f)(x) \vee \chi_A(x) \\ &= \#f \vee \chi_A(x) \\ &= \chi_A(x)\end{aligned}$$

3.1.2 Equivalence relation

Let $x, y \in \emptyset \cup A$ be arbitrarily given. By the definition of `make-sum-comparator`, the following is true:

$$\begin{aligned}x \sim_B y &= (\chi_\emptyset(x) \wedge \chi_\emptyset(y) \wedge x \sim_\emptyset y) \vee (!\chi_\emptyset(x) \wedge !\chi_\emptyset(y) \wedge x \sim_A y) \\ &= (\mathrm{const}_1(\#f)(x) \wedge \mathrm{const}_1(\#f)(y) \wedge x \sim_\emptyset y) \vee (!\mathrm{const}_1(\#f)(x) \wedge !\mathrm{const}_1(\#f)(x) \wedge x \sim_A y) \\ &= (\#f \wedge \#f \wedge x \sim_\emptyset y) \vee (\#t \wedge \#t \wedge x \sim_A y) \\ &= \#f \vee x \sim_A y \\ &= x \sim_A y\end{aligned}$$

3.1.3 Order

Let $x, y \in \emptyset \cup A = A$ be arbitrarily given. By the definition of `make-sum-comparator`, the following is true:

$$\begin{aligned}x <_B y &= (\chi_\emptyset(x) \wedge \chi_\emptyset(y) \wedge x \leq_\emptyset y) \vee (\chi_\emptyset(x) \wedge !\chi_A(y)) \vee (!\chi_\emptyset(x) \wedge !\chi_\emptyset(y) \wedge x \leq_A y) \\ &= (\mathrm{const}_1(\#f)(x) \wedge \mathrm{const}_1(\#f)(y) \wedge x \leq_\emptyset y) \vee (\mathrm{const}_1(\#f)(x) \wedge !\chi_A(y)) \vee (!\mathrm{const}_1(\#f)(x) \wedge !\mathrm{co} \\ &= (\#f \wedge \#f \wedge x \leq_\emptyset y) \vee (\#f \wedge !\chi_A(y)) \vee (!\#f \wedge !\#f \wedge x \leq_A y) \\ &= \#f \vee \#f \vee x \leq_A y \\ &= x \leq_A y\end{aligned}$$

3.1.4 Hashing

This part is left as an exercise to the reader. Much of the same caveats apply as in the case of products.

3.1.5 Conclusion

Therefore, if we ignore the aforementioned caveats, for all comparators $\mathcal{A}$, we have $\mathcal{Q} \oplus \mathcal{A} = \mathcal{A}$.

3.2 Right

The proof for the right case is wholly analogous to the left case.

3.3 Conclusion

The following comparator is a left and right neutral element for $\oplus$ in the example implementation:

$$\mathcal{Q} = (\chi_\emptyset, \sim_\emptyset, <_\emptyset, \#_\emptyset).$$

Furthermore, if we ignore hashing, it is a left right neutral element for $\oplus$ in any conforming implementation.

4 Conclusion

With some caveats related to hashing, we have defined neutral elements for `make-product-comparator` and `make-one-comparator` from srfi 228.

These can easily be implemented in terms of srfi 1 and srfi 128:

```
(define comparator-one
  (make-comparator (const #t)
                   (const #t)
                   (const #f)
                   (const 0)))

(define comparator-zero
  (make-comparator (const #f)
                   (const #f)
```

```
                    (const #t)
                    (const 0)))
```

Or, alternatively, `comparator-zero` can be implemented as follows (where `error` is a function that takes any arguments and raises an error):

```
(define comparator-zero
  (make-comparator (const #f)
                   error
                   error
                   error))
```

I wrote this document at a fairly rapid pace, and I have not proof-read it before submitting it, so I'm not sure whether it's correct in all details. I am, however, convinced that is at the very least morally correct, and I hope I've convinced the reader of this notion.