

Using Iris for Program Verification

Abel Nieto

March 18, 2020

1 Monotone Counter

▷ **Coq code:** <https://github.com/abeln/iris-practice/blob/master/counter.v> ◁

This is the counter example from Section 7.7 of the notes. The module exports three methods:

- **newCounter** creates a new counter. Counters are represented simply with an int reference.

```
Definition newCounter : val := \lambda: <>, ref #0.
```

- **read** returns the value currently stored in a counter.

```
Definition read : val := \lambda: "c", !"c".
```

- **incr** takes a counter, increments it, and returns unit.

```
Definition incr : val :=  
  rec: "incr" "c" :=  
    let: "n" := !"c" in  
    let: "m" := #1 + "n" in  
    if: CAS "c" "n" "m" then #() else "incr" "c".
```

The client for the counter is a program that instantiates a counter, spawns two threads each incrementing the counter, and then reads the value off the counter:

```
Definition client : val :=  
  \lambda: <>,  
    let: "c" := newCounter #() in  
    ((incr "c") ||| (incr "c")) ;;  
    read "c".
```

1.1 Specs

We can prove two different specs for the code above:

- Authoritative RA

In the first spec, our resource algebra is $\text{AUTH}(\mathbb{N})$. Elements of this RA are of the form either $\bullet n$ (authoritative) or $\circ n$ (non-authoritative), where $n \in \mathbb{N}$.

The important properties of this RA (for the counter example) are:

- $\bullet 0 \cdot \circ 0 \in \mathcal{V}$
- $\bullet m \cdot \circ n \in \mathcal{V}$ implies $m \geq n$
- $\bullet m \cdot \circ n \rightsquigarrow \bullet(m+1) \cdot \circ(n+1)$
- $|\circ n| = \circ n$

Then we can define our predicate for the counter:

$$\text{isCounter}(l, n, \gamma) = \ulcorner \circ \bar{n} \urcorner^\gamma * \exists s. \boxed{\exists m. l \hookrightarrow m \wedge \ulcorner \bullet \bar{m} \urcorner^\gamma}^s$$

This predicate is persistent because the invariant component is persistent and $\circ n$ is also persistent (via the **PERSISTENTLY-CORE** rule), because the core of $\circ n$ is $\circ n$ itself.

The intuition behind the predicate is that if we own $\text{isCounter}(l, n, \gamma)$ then, at an atomic step (e.g. a load), we can know that l is allocated and it points to a value that's $\geq n$.

Additionally, after an increment, we can update the resource to $\text{isCounter}(l, n+1, \gamma)$.

We then prove the following specs for the counter methods:

- $\{\top\} \text{newCounter } () \{l. \exists \gamma. \text{isCounter}(l, 0, \gamma)\}$
- $\{\text{isCounter}(l, n, \gamma)\} \text{read } l \{v. v \geq n\}$. Notice how we don't have to give the counter back, because it's persistent.
- $\{\text{isCounter}(l, n, \gamma)\} \text{incr } l \{\text{isCounter}(l, n+1, \gamma)\}$

We can also prove the client spec:

$$\{\top\} \text{client } () \{n. n \geq 1\}$$

The “problem” with this specification *isCounter* doesn't tell us anything about what other threads are doing with the counter. So in the verification of the client code, after both threads return, we know

$$\text{isCounter}(l, 1, \gamma) * \text{isCounter}(l, 1, \gamma) \equiv \text{isCounter}(l, 1, \gamma)$$

So we know that the counter must be ≥ 1 , but we don't know that it's 2.

- Authoritative RA + Fractions

The second spec we can give is more precise, and allows us to conclude that the client reads exactly 2 once the two threads finish.

The resource algebra we use is $\text{AUTH}((\mathbb{Q}_p \times \mathbb{N})_?)$:

- \mathbb{Q}_p is the RA of positive fractions, where the valid elements are those ≤ 1 .
- The $?$ is the *optional RA* construction. In this case, it's needed because to use the authoritative RA we need the argument algebra to be *unital*. And $\mathbb{Q}_p \times \mathbb{N}$ is not unital because 0 is not an element of \mathbb{Q}_p .

The important properties of this RA are:

- $\circ(1, 0) \cdot \bullet(1, 0) \in \mathcal{V}$
- $\circ(p, n) \cdot \bullet(1, m) \in \mathcal{V}$ implies $m \geq n$
- $\circ(1, n) \cdot \bullet(1, m) \in \mathcal{V}$ implies $m = n$
- $\circ(p, n) \cdot \bullet(q, m) \rightsquigarrow \circ(p, n+1) \cdot \bullet(q, m+1)$

With this RA, we get the new counter resource

$$\text{isCounter}(l, n, p, \gamma) = \left[\begin{array}{c} \text{---} \\ \circ(p, n) \\ \text{---} \end{array} \right]^\gamma * \exists s. \left[\begin{array}{c} \text{---} \\ \exists m. l \hookrightarrow m \wedge \bullet(1, m) \\ \text{---} \end{array} \right]^\gamma \Bigg]^s$$

This resource is no longer persistent, because the core $|p|$ is undefined. However, we can show the following

$$\text{isCounter}(l, n+m, p+q, \gamma) \dashv\vdash \text{isCounter}(l, n, p, \gamma) * \text{isCounter}(l, m, q, \gamma)$$