CNode Allocation

Dan Pittman dan@auxon.io

September 10, 2021

Slot Ranges

CSpace slots in Ferros are allocated as SlotRanges. A SlotRange is a contiguous range of slots in a CNode. The reader should note that CSpaces in Ferros are flat. That is, Ferros does not provide a manner with which to insert a CNode into a CNode. Therefore, CSpace slot allocation is flat, and can be done via contiguous ranges of slots.

```
record SlotRange (sz : ℕ) : Set where
  field
    offset : ℕ

open SlotRange

sr-size : {x : ℕ} → SlotRange x → ℕ
sr-size {x = x} _ = x
```

SlotRange is the datatype which corresponds to the type with the same name in Ferros:

```
pub struct CNodeSlots<Size: Unsigned, Role: CNodeRole> {
    pub(super) cptr: usize,
    pub(super) offset: usize,
    pub(super) _size: PhantomData<Size>,
    pub(super) _role: PhantomData<Role>,
}
```

And the function on SlotRange which we with to prove things about is alloc:

```
pub fn alloc<Count: Unsigned>(
    self,
) -> (CNodeSlots<Count, Role>, CNodeSlots<Diff<Size, Count>, Role>)
where
    Size: Sub<Count>,
    Diff<Size, Count>: Unsigned,
{
```

```
    (
        CNodeSlots {
            cptr: self.cptr,
            offset: self.offset,
            _size: PhantomData,
            _role: PhantomData,
        },
        CNodeSlots {
            cptr: self.cptr,
            offset: self.offset + Size::USIZE,
            _size: PhantomData,
            _role: PhantomData,
        },
    )
}
```

Its Agda definition is as follows:

```
alloc : {isz : ℕ} →
        SlotRange isz →
        (count : ℕ) →
        (p : count ≤ isz) →
        SlotRange (ℕ-sub isz count p) × SlotRange count
alloc sr count _ =
  let os = offset sr
  in record { offset = os } , record { offset = os + count }
```

alloc requires a proof that the `count`, that is the size of the range which is to be allocated from the initial range, is less than or equal to that of the initial range. It is through this proof that we can guarantee that a range can be allocated. Also note that this proof, which would appear in Rust as a type bound, is implied by the definition of `Diff`, however in Agda we must be explicit.

Now we intend to prove that when we allocate a range from another that nothing is lost. This is done through a simple arithmetic proof which cancels the subtraction by adding the LHS operad back to the result which should be equal to the size of the initial range.

```
alloc-retains : ∀ (x count : ℕ) →
                (p : count ≤ x) →
                (sr : SlotRange x) →
                sr-size (proj_1 (alloc sr count p)) + sr-size (proj_2 (alloc sr count p)) ≡ x
alloc-retains x count p _ = invert-ℕ-sub x count p
```

This of course implies, though the symmetric property of equality, that we can reconstruct a range through the sum of its parts.

```
alloc-retains-reconstruct : ∀ (x count : ℕ) →
  (p : count ≤ x) →
  (sr : SlotRange x) →
  x ≡ sr-size (proj_1 (alloc sr count p)) + sr-size (proj_2 (alloc sr count p))
alloc-retains-reconstruct x count p sr = sym (alloc-retains x count p sr)
```