Record operations in Cogent

September 8, 2020

1 Permissions

A type may have multiple permissions among S (sharable), D (discardable), E (escapable). Sharable elements of the context can be duplicated, whereas discardable elements can be dropped.

We deem **non-linear** any type which is sharable and discardable. Any type can become non-linear through the bang type operation.

We call **writable** a type which allows in-place modifications. For example, unboxed types are not writable in this sense, but writable records are.

Remark 1. *Currently, any sharable or discardable type is non linear. However, the distinction between sharable and discardable is maintained in the view of future extensions.*

Remark 2. *Non-linear types currently coincide with non-writable (in the sense of in-place updatable) types. This coincidence is crucial for the typing rule of Member, as we will see.*

2 Records

Each record type specifies a mode r (read-only), w (writable), u (unboxed). The permissions of a record type are a subset of the permissions of its mode, recalled below:

$$r :_\kappa D, S \qquad w :_\kappa E \qquad u :_\kappa D, S, E$$

More exactly, a record type has some permission if its mode has it and each (non-taken) field type has it (Cf. rule KRec of the ICFP paper). Banging w yields r.

2.1 Taken fields

In order to enforce linearity, the type system keeps track of which fields have been already taken in a record (in order to prevent accessing twice the same field). Vincent noted however that it is unnecessary to keep track of this information for the subfields of a taken field.

2.2 Operations on records

The record operations consist in updating or getting some field.

The most obvious get operation is called member: it returns the field value of some record. However, this member operation is too restrictive: it does not allow to access the different fields of a non sharable record. This is the motivation for the alternative Take operation, which returns (in some sense) the record.

2.2.1 Member

The member operation allow to access a (present) field of a sharable record.

Critique This typing rule does not match Liam's PhD thesis in two respects:

- it involves the sharable kind rather than discardable kind (although currently both are equivalent, see Remark 1)
- the whole record is required to have the kind, rather than the record without the involved field. The reason for enforcing that the involved field has the kind is, I guess, to prevent accessing a writable field of a readonly record[1]. However, this relies on the coincidence that in the current system, non-linear types coincide with non-writable types (Remark 2), which might not be true in the future.

Suggestions First, we could require that the record without the involved field is discardable. Then, this opens the possibility of accessing a writable field of a readonly record. Here are some ways of forbiding this, in the order of increasing required amount of work (according to me):

1. introduce a well-formed predicate on types to prevent nesting a writable record in a readonly record[2].
2. Introduce a new kind W for writable types, and forbids accessing a writable field of a readonly record.

Remark 3. *Another suggestion consists in banging the return type of member when the record is readonly. However, Vincent is concerned that this does not match the bang supposed behaviour, which might be problematic for future extensions. I trust him.*

[1] This is unwanted because then the writable subrecord could be in-place updated, although it was part of a readonly record.

[2] Although such types are constructible in the ICFP paper, no operation is available for them according to the ICFP paper, and they are not constructible with the implemented surface language.

2.3 Take

access a (present) field of a non-read-only record, with a continuation where the field is marked as taken. In the case the field is sharable, this marking is not necessary, although still possible.

Remark 4. *In these remarks, I assume that Member has the corrected typing rule suggested above.*

- *Instead of having a continuation, Take could return the field and the record with the field marked as Taken.*

Critique Member can be thought of as a simplified version of Take. Its status is not very clear, because in some specific case it is redundant with Take, and in others it is not.

Suggestion It would make sense to extend Take so that it covers the Member case as well. Then Member could be thought merely as an additionnal convenience. In this case, we just have to remove the non-readonly constraint (in this case, some, we should prevent accessing a writable field of a read-only record, as following suggestions of the previous section on Member).

2.3.1 Put

updating a taken or[3] discardable field of a non-read-only record, returning the updated record where the field is marked as present.

2.3.2 Remarks

Remark 5. *As mentioned above, read-only records with at least one non-sharable field are useless, as no operation is available for them, although it would be harmeless to allow take or member for their sharable fields.*

Remark 6. *One may consider allowing Take and Member for sharable taken fields. This would be safe if taken were not used to mark uninitialized fields*[4]*. But ignoring this issue (for example, introducing a proper uninitialized flag), one could then consider removing the possibility of keeping a sharable field as present for Take. But, in this case, you could not take a sharable field and return the original record without taken annotation, so some subtyping rule should be added to make the taken annotation for sharable fields irrelevant, or, as a hack, you could re-put the taken value (or introduce an explicit cast operation).*

[3] This 'or' is not exclusive.

[4] The return type of abstract functions can contain taken annotations, meaning that some fields are not initialized.