Specifications and private data: A call for privatizable variables

K. Rustan M. Leino

6 September 1994

This note takes a look at specifications surrounding "private data", and suggests a notion of *privatizable* variables. We assume a specification language like [Bolt] and the ideas from [KRML40].

0 The given specifications

Consider the following specification of an interface `A`.

```
INTERFACE A;

TYPE T <: REFANY;
SPEC VAR valid: T → BOOLEAN;
SPEC VAR state: T → ANY;

PROCEDURE init(t: T);
  MODIFIES state[t], valid[t]
  ENSURES valid[t]

PROCEDURE update(t: T);
  REQUIRES valid[t]
  MODIFIES state[t]

PROCEDURE destroy(t: T);
  MODIFIES state[t], valid[t]
```

Also consider a friends interface `AFriends`.

```
INTERFACE AFriends;
IMPORT A;

VAR previous: A.T;
DEPENDS A.state → previous;
DEPENDS A.valid → previous;
```

Now consider `B`, whose interface is as follows.

```
INTERFACE B;

TYPE T <: REFANY;
SPEC VAR valid: T → BOOLEAN;
SPEC VAR state: T → ANY;

PROCEDURE init(t: T);
  MODIFIES state[t], valid[t]
  ENSURES valid[t]

PROCEDURE update(t: T);
  REQUIRES valid[t]
  MODIFIES state[t]
```

The implementation of `B` reveals the details of type `B.T`.

```
MODULE B;
IMPORT A;

REVEAL T = BRANDED OBJECT
    x: PRIVATE A.T
  END;
```

Notice that `B` is a regular client of `A`, and thus does not import `AFriends`.

The keyword `PRIVATE` indicates that field `x` is never "imported" into or "leaked" from this module. The motivation for this is as follows. The state and validity of a `B.T` object depends on the state and validity of the fields of a `B.T`. Thus, `B` gives the following information about its specification variables.

```
DEPENDS valid[t: T] → A.valid[t.x];
DEPENDS state[t: T] → A.state[t.x];
REP valid[t: T]: valid[t] = A.valid[t.x];
```

But this violates the Visibility and Authenticity requirements from [KRML40]. At this point, our feeling is that the violation is for silly reasons; if `B.T` happens to use an `A.T` in its private implementation, why does this need to be advertised in `A`, which knows nothing about `B`? The `PRIVATE` keyword used in conjunction with the declaration of field `x` is an attempt at allowing the implementation to depart from the Visibility and Authenticity requirements. The details of `PRIVATE` are still under experimentation.

Module `B` also includes the implementation of the procedures in interface `B`.

```
PROCEDURE init(t: T) =
  A.init(t.x)

PROCEDURE update(t: T) =
  A.update(t.x)
```

1 A bad use of A and B

Let's take a look at how a client may use `A` and `B`.

```
MODULE Client;
IMPORT A, AFriends, B;

PROCEDURE P()
  MODIFIES A.state, A.valid, B.valid, B.state
=
  VAR b := NEW(B.T); BEGIN
    B.init(b);
    IF AFriends.previous # NIL THEN A.destroy(AFriends.previous) END;
    B.update(b)
  END
```

The question here is, can `P` meet the precondition of `B.update`. From the information given, and the rules from [KRML40], the verification would indeed validate `P` as having the required precondition of `B.update`. But consider an implementation of `A.init(t)` that, in addition to initializing `t`, sets `AFriends.previous` to `t`. Then we do *not* want the verification of procedure `P` to validate, because there is no guarantee that the call to `B.update` meets its precondition.

The problem seems to be that although `B` promised not to leak a value of a `B.T.x` field, `B` passed such a value to the procedures of `A`, and `A` never made a promise not to leak such values. Hence, we don't want `B` to get away with declaring `B.T.x PRIVATE`, unless `A` guarantees that instances of `A.T` are *privatizable*.

Let us be more precise. We are discussing the circumstances under which the declaration of dependents of a specification variable are allowed to depart from the Visibility and Authenticity requirements. In the above, in order for `B.valid[t]` to be declared to depend on `A.valid[t]`, the latter specification variable must have been declared to be privatizable. If `A.valid[t]` is privatizable, its potential dependencies are restricted. A research goal is to find a good set of rules for this. For example, one may require that `A.valid[t]` must only depend on entities of the form `w[t]` where `w[t]` is also privatizable, and a field `w` of type `T` is, as a base case, given to be privatizable.

2 Private data and NEW

To show a related issue, consider the following interface `S` and module `M`.

```
INTERFACE S;
PROCEDURE kip();
  MODIFIES (* nothing *)


MODULE M;
IMPORT S, AFriends;
```

```
PROCEDURE P() =
  VAR p := AFriends.previous; BEGIN
    S.kip();
    IF p # AFriends.previous THEN Wrong END
  END
```

Given the above, and the rules from [KRML40], the verification will validate `P`. However, consider the following implementation of `S.kip`.

```
MODULE S;
IMPORT A;

PROCEDURE kip() =
  VAR a := NEW(A.T); BEGIN
    A.init(a)
  END
```

If this is indeed the implementation of `S.kip`, a validation of `M.P` would not be sound.

One view of the world is that since the variable `a` was just allocated in `S.kip`, any modification that it may undergo is okay. Since there's nothing interesting to do with a reference itself, one might interpret this view of the world as saying that a procedure may modify anything of the form `w[r]`, for any reference `r` that was not allocated on entry to the procedure.

This view of the world is flawed, unless one says more about `w`. It seems reasonable to say that `w[r]` must be privatizable.

3 Conclusion

This note carries "privatizable variables" in its title. However, it is not certain that it is the variables that need to be privatizable. Rather, it may be types, procedures, or even interfaces. Since we have seen a problem with both private data fields of a type and data local (private) to a procedure, optimism has us rooting for privatizing variables, since such a solution, if a solution at all, would cover both types and procedures.

References

[Bolt] D. Detlefs and G. Nelson. *Bolt: The Esc Specification Language*. DEC SRC, 1994.

[KRML40] K.R.M. Leino and G. Nelson. *With Scopes in Mind: Abstraction Functions in Specifications*. August 1994.