# OCAPs – London meeting December 2015

## 1. MayRead – a new predicate

We need a new predicate which says that o may read a field f or another object o'. This is different

We have three possibilities

$M, \kappa \vDash MayRead_1(x,o.f)$

Iff

x may call a method whose execution contains the value of o.f

(ie exists a method m, so that M, κ, x.m(….) ↝ _,_ and the execution "mentions" the value of o.f as in κ.

To describe this, we will need to instrument the definition of ↝ with appropriate effects to talk about "mentions".

$M, \kappa \vDash MayRead_2(x,o.f)$

Iff

x may call a method whose execution will have the effect of reading the field f from o

(ie exists a method m, so that M, κ, x.m(….) ↝ _,_ and the execution "reaches" o, and then reads its field f.

To describe this, we will need to instrument the definition of ↝ with appropriate effects to talk about reading a field f from address o.

$M, \kappa \vDash MayRead_3(x,o.f)$

Iff

x may may "observe" the value of o.f

(ie there exists a value v, and a method m, so that

M, κ, x.m(….) ↝ v1,_ and M, κ[o.f↦v], x.m(….) ↝ v',_ and v1=/=v2

This is a bit like information flow.

NOTES

1) $MayRead_1$ has the disadvantage of "what if the execution had the value independently"
2) $MayRead_2$ cannot distribguish between partuial (and perhaps inccuous access the to the fied, eg a service reads the age of the reveiver in order to send some wine, but does not disclose it to the sender)
3) $MayRead_3$ has the same weakness as $MayRead_2$ – this is what we wrote, but SD does not agree any more.

## 2. Specification of Sealer/Unsealer

**specification** SU{

Pol_1:
**true**
{ res= this.make() }
res ***obeys*** Secret ∧ res **isFresh**

Pol_2:
**true**
{ res= this.seal(o,secret) }
res ***obeys*** Envelope
∧ Sealed(o,secret,res)
∧ ∀o. MayAccess(o,obj) ⇒. $\text{MayAccess}_{PRE}$(o,obj)
∧ ∀o. MayAccess(o,secret) ⇒. $\text{MayAccess}_{PRE}$(o,secret)
// the two above ensure that the SU does not "leak" obj, nor secret

Pol_3:
Sealed(obj,secret,envelope)
{ **anyCode** }
∀o. MayAccess(o,obj) ⇒.
$\text{MayAccessPublicTrans}_{PRE}$(o,obj)
∨ ( $\text{MayAccessPublicTrans}_{PRE}$(o,envelope) ∧ $\text{MayAccessPublicTrans}_{PRE}$(o,secret) )
// the above is abut defensive consistency; it ensures, eg that SU does not give a further
// function (say cheat()) which leaks obj

Pol_4:
Sealed(obj,secret,envelope)
{ **anyCode** }
∀o. MayAccess(o,secret) ⇒. $\text{MayAccessPublicTrans}_{PRE}$(o,secret)
// the above is also about defensive consistency; it ensures, eg that SU does not give a further
// function (say cheat()) which leaks obj
}

NOTES

1) We need to clarify the "Public" part in the predicate MayAccessPublicTrans.
2) We also need to clarify the Trans part in the MayAccessPublicTrans, namely, if o1 may access o3, and o2 may access o4 and o5, then MayAccessPublicTrans(o1,o5) and also MayAccessPublicTrans(o1,o5), because through method calls of the intermediate object, o1 may get access to o5, and o5 may get access to o1.

**specification** Envelope{

**ghost** cnts, scrt

Pol_10:

**true**

{ res = this.unseal(secret} }

res∈Obj ⇔ $Sealed_{PRE}$(obj,secret,envelope)

res∈Obj ⇒. res = obj

∀o. MayAccess(o,secret) ⇒. $MayAccessPublicTrans_{PRE}$(o,secret)

// should we add NOT(Sealed (obj,secret,envelope))?

// Note that we did not use the ghostfields0

∀o,s. s ***obeys*** Secret ∧ MayAccess(o,s) ⇒. $MayAccesss_{PRE}$(o,s)

Pol_11:

Sealed (obj,secret,envelope) ⇔ envelope.cnts=obj ∧ envelope.secret=scrt

Pol_12:

this.scrt **obeys** Secret

Pol_13:

∀o. $MayRead_2$(o,this.cnts) ⇒. MayAccessPublicTrans (o,this.scrt)

// The above fobids getting access to the cnts of an envelope unless we can also get

// hold of the secret.

// However, it does not preclude Envelope to offer further methods what return cnts,

// providedthe caller of the method can also provide the secret.?

}

NOTES

1) In the above, MayRead_2 stands for the 2nd version from our notes on the board, where it means that o
2) Discuss difference Pol_4 and Pol_13