Permission and Authority Revisited towards a formalisation

Sophia Drossopoulou
Imperial College London
scd@doc.ic.ac.uk

James Noble
Victoria University Wellington
kjx@ecs.vuw.ac.nz

Mark S. Miller
Google Inc.
erights@google.com

Toby Murray
The University of Melbourne
toby.murray@unimelb.edu.au

ABSTRACT

Miller's notions of *permissions* and *authority* are foundational to the analysis of object-capability programming. Informal definitions of these concepts were given in Miller's thesis. In this paper we propose precise definitions for permissions and authority, based on a small object-oriented calculus. We quantify their bounds (current, eventual, behavioural, topological), and delineate the relationships between these definitions.

CCS Concepts

•Security and privacy → Logic and verification; •Software and its engineering → Semantics;

Keywords

Permission; Authority; Object-Capabilities.

1. INTRODUCTION

In his doctoral dissertation [11], Mark Miller proposed the concepts of *permissions* and *authority* of an object *o* to describe which other objects *o* has access to, and which other objects it may modify. Miller also distinguished between four flavours of authority: the *current*, *eventual*, *behavioural*, and *topological* bounds on these permissions and authorities.

The concept of authority has been used to verify or analyse object-capability systems [10, 12], however, to our knowledge, there exists no work which concentrates on the foundational formal definition of these concepts and their flavours. Such a foundational definition is required as object-capability systems and their formalisations are increasingly of interest in the research literature [6, 2]. However, these works tend to use only one or two these concepts, as needed for the particular study, and the particular programming language. To our knowledge, there does not exist a study of *all* flavours of permission and authority, and their relationship, in a language-independent setting.

FTfJP'16, July 19 2016, Rome, Italy

ISBN 978-1-4503-4439-5/16/07…$15.00

DOI: http://dx.doi.org/10.1145/2955811.2955821

In this paper we propose precise definitions for *all* these concepts, and discuss their relationships. We base the model on a small class-based object-oriented calculus. In the process of our formalisation, we uncover one new category of authority, we formalise some of the claims made in [11], and rectify some others. The definitions we propose are precise, but do not necessarily reflect all aspects relevant to permissions and authority. In the last section of our paper we discuss some aspects that we have not yet been able to cover, and outline how the definitions could be improved.

2. FORMAL PRELIMINARIES

We assume a program P written in a memory safe OO language with no ambient authority (i.e. no global or static variables, so that objects can only interact with other objects which they have been passed explicitly). We assume a small step operational semantics of the shape

$$P \vdash \sigma, \texttt{stmts} \rightsquigarrow \sigma', \texttt{sr},$$

where σ is the runtime state, `stmts` are statements, and `sr` is either statements or a result (the address of an object, $null$, a number etc).

Figure 1 gives an example of such a language, which however serves illustration purposes only. Similar concepts as those we define below should be definable in any other such programming languages. The most important feature of this core language is that fields are encapsulated: as in Smalltalk, the syntax enforces that fields may be read or written only by the object to which they belong.

We have not allowed for loops and conditionals. These can easily be encoded. We give the definition of states, σ, in figure 2 and operational semantics in figure 3. The runtime states are defined in terms of concepts are as found in many related works: the state σ consists of a sequence of stack entries, ψs; each ψ consists of a frame ϕ, mapping from variable identifiers to addresses in the heap, and a continuation $C[\circ]$, indicating how execution will continue when the callee has terminated.

We define *liberal execution* of the shape

$$\sigma \overset{?}{\rightsquigarrow} \sigma',$$

which describes all *possible executions* within the programming language, for all programs P and statements `stmts`.

DEFINITION 1. *For any program P, if $P \vdash \sigma, \texttt{stmts} \rightsquigarrow \sigma', \texttt{sr}$ then $\sigma \overset{?}{\rightsquigarrow} \sigma'$. And, if $\sigma \overset{?}{\rightsquigarrow} \sigma'$ then there exists program P and statements `stmts` and result `sr` such that $P \vdash \sigma, \texttt{stmts} \rightsquigarrow \sigma', \texttt{sr}$.*

That is: $\sigma \overset{?}{\rightsquigarrow} \sigma'$ if we can imagine a program P and statements `stmts` that somehow get us to σ'.

Program	::=	*ClassDescr**
ClassDescr	::=	**class** *ClassId*
		{ (**field** *FieldId*)* (*methBody*)* **}**
methBody	::=	**method** *m* (*ParId**) **{** *Stmts* **}**
Stmts	::=	*Stmt* | *Stmt* **;** *Stmts*
Stmt	::=	*ParId* **:=** *Rhs*
	|	**this**.*FieldId* **:=** *Rhs*
	|	**return** *Rhs*
Rhs	::=	*Arg*.*MethId*(*Arg**) | *Arg*
	|	**new** *ClassId*(*Arg**)
Arg	::=	**this** | *ParId* | **this.** *FieldId*
	|	**true** | **false** | **null**

$ClassId, ParId, MethId, VarId, FieldId \in Identifier$

Figure 1: Syntax

$\sigma \in$ *state*	::=	$\psi s \cdot \phi \cdot \chi$
ψs	::=	ψ^*
ψ	::=	$\phi \cdot C[\circ]$
$\phi \in$ *frame*	=	*StackId* $\longrightarrow$ *val*
$C[\circ]$	::=	$ParId := \circ$ | $\mathbf{this}.FieldId := \circ$
	|	**return** $\circ$ | $C[\circ]; Stmts$
$\chi \in$ *heap*	=	*addr* $\longrightarrow$ *object*
$\mathtt{v} \in$ *val*	=	$\{$ `null`, **true**, **false** $\} \cup addr$
object	=	*ClassId* $\times$ (*FieldId* $\longrightarrow$ *val*)
$\iota, \iota', ..$	$\in$	*Address*
StackId	=	$\{$ **this** $\} \cup$ *VarId* $\cup$ *ParId*

Figure 2: States

We also assume a *behavioural description* B of execution which describes a set of permissible executions with the shape

$$B \vdash \sigma, \mathtt{stmts} \rightsquigarrow \sigma', \mathtt{sr},$$

such that $B, \sigma, \mathtt{stmts} \rightsquigarrow \sigma', \mathtt{sr}$ implies that $\sigma \overset{?}{\rightsquigarrow} \sigma'$. A behavioural description may be anything ranging from a complete functional specification, e.g., [8], or low-level mechanisms such as owners-as-dominators [1], owners-as-modifiers [9], or partial specification of particular policies as advocated in [3, 4]. We say that a behavioural description characterises a program if all the executions of the program fall within the behavioural description:

DEFINITION 2. *$P \subset B$ iff*
$\forall \sigma, \sigma', \mathtt{stmts}, \mathtt{sr}.$
$P \vdash \sigma, \mathtt{stmts} \rightsquigarrow \sigma', \mathtt{sr}$ *implies* $B \vdash \sigma, \mathtt{stmts} \rightsquigarrow \sigma', \mathtt{sr}$

In other words, $P \subset B$ means that P is a refinement of B.

These three semantics give rise to three definitions of possible future worlds. The *Eventual Worlds* (EW) are those states σ' reachable from state σ by a program P; the *Behavioural Words* (BW) are those states reachable under a behavioural description B; and the *Maximal Worlds* (MW) are those reachable via liberal execution. (MW corresponds most closely to Miller's "Topology-based" bounds).

DEFINITION 3.

$$EW(P,\sigma) \equiv \{\, \sigma' \mid \exists \mathtt{stms}, \mathtt{sr}.\ P \vdash \sigma, \mathtt{stmts} \rightsquigarrow^* \sigma', \mathtt{sr} \,\}$$

$$BW(B,\sigma) \equiv \{\, \sigma' \mid \exists \mathtt{stms}, \mathtt{sr}.\ B \vdash \sigma, \mathtt{stmts} \rightsquigarrow^* \sigma', \mathtt{sr} \,\}$$

$$MW(\sigma) \equiv \{\, \sigma' \mid \sigma \overset{?}{\rightsquigarrow}{}^* \sigma' \,\}$$

It is easy to see that the following inclusions hold between the current, eventual, behavioural, and maximal future worlds:

LEMMA 1. *For all programs P, behavioural description B, and states σ:*
- $\sigma \subseteq EW(P,\sigma) \subseteq MW(\sigma)$
- $BW(B,\sigma) \subseteq MW(\sigma)$
- *If* $P \subset B$ *then* $EW(P,\sigma) \subseteq BW(B,\sigma)$

Moreover, it is easy to show that with increasing stack prefixes (i.e., increasing the method calls in the calling context of the current configuration σ), we increase the worlds:

LEMMA 2. *For all programs P, behavioural description B, states σ, and stack prefixes ψs:*
- $EW(P,\sigma) \subseteq EW(P,\psi s \cdot \sigma)$
- $BW(B,\sigma) \subseteq BW(B,\psi s \cdot \sigma)$
- $MW(\sigma) \subseteq MW(\psi s \cdot \sigma)$

3. FOUR FLAVOURS OF PERMISSIONS

Citing Lampson [7], Miller [11] defines permission in an object-capability system: "A direct access right to an object gives a subject the permission to invoke the behaviour of that object". More precisely, we define that an object o's *Current Permissions* (CP) in a state σ are simply all the object references to which o has direct access in σ: the contents of o's fields `f`. In addition, if o is the current receiver, then CP includes the local variables `x` to which the current method has access.

DEFINITION 4.

$$CP(\sigma,o) \equiv \{\, o \,\} \cup \{\, o' \mid \exists \mathtt{f}.\ \sigma(o,\mathtt{f}) = o' \,\} \cup \{o' \mid o = \sigma(\mathtt{this}) \wedge \exists \mathtt{x}.\ \sigma(\mathtt{x}) = o' \,\}$$

In the above $\sigma(\mathtt{this})$ indicates the receiver of the currently executing method, i.e. $\sigma(\mathtt{this})$=$\phi(\mathtt{this})$, when $\sigma = \_ \cdot \phi \cdot \chi$, for some χ.

Then, we can define bounds on future permissions by projecting CP into the future worlds of σ, giving precise definitions of *Eventual Permissions* (EP), *Behavioural Permissions* (BP) and *Maximal Permissions* (MP):

DEFINITION 5.

$$EP(P,\sigma,o) \equiv \bigcup_{\sigma' \in EW(P,\sigma)} CP(\sigma',o) \cap dom(\sigma)$$

$$BP(B,\sigma,o) \equiv \bigcup_{\sigma' \in BW(P,\sigma)} CP(\sigma',o) \cap dom(\sigma)$$

$$MP(\sigma,o) \equiv \bigcup_{\sigma' \in MW(P,\sigma)} CP(\sigma',o) \cap dom(\sigma)$$

Note that each of these definitions projects the permissions back to those objects existing in world σ ($\cap\, dom(\sigma)$) because knowing that more objects will be created in some future world of the current state σ doesn't give us any more useful information about the objects that actually exist in the current state.

The following relations hold:

$\lfloor \mathbf{this} \rfloor_{\phi\cdot\chi} = \phi(\mathbf{this})$ $\quad \lfloor parId \rfloor_{\phi\cdot\chi} = \phi(parId)$ $\quad \lfloor \mathbf{this}.fldId \rfloor_{\phi\cdot\chi} = \chi(\phi(\mathbf{this}), fldId)$

$\lfloor \mathbf{null} \rfloor_{\phi\cdot\chi} = \mathbf{null}$ $\quad \lfloor \mathbf{true} \rfloor_{\phi\cdot\chi} = \mathbf{true}$ $\quad \lfloor \mathbf{false} \rfloor_{\phi\cdot\chi} = \mathbf{false}$

(NEW_OS)

ι is new in χ
$\mathtt{f}_1, ...\mathtt{f}_n$ are the fields for CId as defined in P

$$P \vdash \phi\cdot\chi,\ \mathbf{new}\ CId(\ a_1, ...a_n) \rightsquigarrow\ \chi[\iota \mapsto (CId, \mathtt{f}_1 \mapsto \lfloor a_1 \rfloor_{\phi,\sigma} ... \mathtt{f}_n \mapsto \lfloor a_n \rfloor_{\phi,\sigma})],\ \iota$$

(ARG_OS)

$$P \vdash \phi\cdot\chi,\ a\ \rightsquigarrow\ \chi,\ \lfloor a \rfloor_{\phi,\sigma}$$

(VARASG_OS)

$$\frac{P\ \vdash\ \phi\cdot\chi,\ rhs\ \rightsquigarrow\ \chi',\ val}{P\ \vdash\ \psi s\cdot\phi\cdot\chi,\ v{:=}rhs\ \rightsquigarrow\ \psi s\cdot\phi[v \mapsto val]\cdot\chi',\ val}$$

(FIELDASG_OS)

$$\frac{P, \vdash \phi\cdot\chi,\ rhs\ \rightsquigarrow\ \chi',\ val}{P \vdash \psi s\cdot\phi\cdot\chi,\ \mathbf{this}.f{:=}\ rhs\ \rightsquigarrow\ \psi s\cdot\phi\cdot\chi'[\phi(\mathbf{this}), f \mapsto val],\ val}$$

(SEQUENCE_OS)

$$\frac{P\ \vdash \sigma, stmt\ \rightsquigarrow\ \sigma',\ val}{P\ \vdash \sigma,\ stmt; stmts\ \rightsquigarrow\ \sigma',\ stmts}$$

(METHCALL_OS)

$\lfloor a \rfloor_{\phi\cdot\chi} = \iota$
$\lfloor a_i \rfloor_{\phi\cdot\chi} = val_i \quad \forall i \in \{1..n\}$
$\mathcal{M}(P, \chi(\iota)\downarrow_1, \mathtt{m}) = \mathbf{method}\ m(\ par_1, \ldots par_n)\ \{\ stms\}$
$\phi' = \mathbf{this} \mapsto \iota, par_1 \mapsto val_1, \ldots par_n \mapsto val_n$

$$P \vdash \psi s\cdot\phi\cdot\chi,\ C[a.m(\ a_1, ...a_n)\ ]\ \rightsquigarrow\ \psi s\cdot\phi\cdot C[\circ]\cdot\phi'\cdot\chi, stms$$

(METHRETURN_OS)

r free in ϕ
$\phi' = \phi[r \mapsto \lfloor a \rfloor_{\phi\cdot\chi}]$

$$P \vdash \psi s\cdot\phi\cdot C[\circ]\cdot\chi,\ \mathbf{return}\, a\ \rightsquigarrow\ \psi s\cdot\phi'\cdot\chi,\ C[r]$$

Figure 3: Operational Semantics

LEMMA 3. *For all programs P, behavioural description B, and states σ:*

- $CP(\sigma, o)\ \subseteq\ EP(P, \sigma, o) \subseteq\ MP(\sigma, o)$
- $CP(\sigma, o)\ \subseteq\ BP(B, \sigma, o) \subseteq\ MP(\sigma, o)$
- *If* $P \subset B$ *then* $EP(P, \sigma, o) \subseteq BP(B, \sigma, o)$

The proof of this lemma follows from lemma 1.
Note the condition in the relationship between behavioural and eventual permissions: the eventual permissions are bounded by the behavioural permissions only when the behaviour of the program P is bounded by the behavioural description B — this condition was omitted from Miller's original definition [11, p.60].

Based on lemma 2 we can prove that by increasing the stack prefix, we increase the eventual, behavioural, and maximal permissions, but not the current ones.

LEMMA 4. *For all programs P, behavioural description B, states σ, and stack prefixes ψs:*

- $CP(\sigma, o) = CP(\psi s\cdot\sigma)$
- $EP(P, \sigma, o) \subseteq EP(P, \psi s\cdot\sigma, o)$
- $BP(B, \sigma, o) \subseteq BP(B, \psi s\cdot\sigma, o)$
- $MP(\sigma, o) \subseteq MW(\psi s\cdot\sigma, o)$

4. FOUR FLAVOURS OF AUTHORITY

Miller [11, p.59] defines authority as "the Ability to Cause Effects". Following the scheme established for permissions, we can define the *Current Authority* (CA) of an object o in state σ of program P:

DEFINITION 6.

$$\begin{aligned} CA(P, \sigma, o)\ \equiv\ \{\ o'\ |\ \exists \sigma'', \mathtt{m}, o_1, ...o_n.\ & \\ o \in CP(\sigma, \sigma(\mathtt{this}))\ \wedge\ & \\ \forall i \in \{1..n\}.o_i \in CP(\sigma, o)\ \wedge\ & \\ P\ \vdash\ \sigma'', x.\mathtt{m}(x_1, ...x_n) \rightsquigarrow^*\ \sigma', \_\ \wedge\ & \\ \sigma'' = \sigma[x \mapsto o, x_1 \mapsto o_1, ...x_n \mapsto o_n]\ \wedge\ & \\ \sigma(o', \mathtt{f}) \neq \sigma'(o', \mathtt{f})\ \} & \end{aligned}$$

Our definition states that the current authority of o in state σ includes all objects o' which may be modified by a method call provided that the method call is possible from the current receiver (i.e. o is reachable from the receiver — $CP(\sigma, o)$), and that it can originate from o. A method call may originate from o if the receiver and arguments of that method are accessible from o (i.e. are in $CP(\sigma, o)$). The concept of CA was not proposed in Miller's thesis [11], which proposed only *Eventual Authority* (EA), *Behavioural Authority* (BA), and *Maximal Authority* (MA). We define these authorities in terms of CA by projecting CA into the three future worlds, in the same way we projected permissions:

DEFINITION 7.

$$EA(P, \sigma, o)\ \equiv\ \bigcup_{\sigma' \in EW(P,\sigma)} CA(\sigma', o)\ \cap\ dom(\sigma)$$

$$BA(B, \sigma, o)\ \equiv\ \bigcup_{\sigma' \in BW(P,\sigma)} CA(\sigma', o)\ \cap\ dom(\sigma)$$

$$MA(\sigma, o)\ \equiv\ \bigcup_{\sigma' \in MW(P,\sigma)} CA(\sigma', o)\ \cap\ dom(\sigma)$$

The following relations hold:

LEMMA 5. *For all programs P, behavioural description B, and states σ:*

- $CA(\sigma, o)\ \subseteq\ EA(P, \sigma, o) \subseteq\ MA(\sigma, o)$

- $CA(\sigma, o) \subseteq BA(B, \sigma, o) \subseteq MA(\sigma, o)$
- *If* $P \subset B$ *then* $EA(P, \sigma, o) \subseteq BA(B, \sigma, o)$

The proof of this lemma also follows from lemma 1.
Note again that the relationship between EA and BA depends on the program conforming to the behavioural description.

Again, based on lemma 2 we can prove that by increasing the stack prefix, we increase the eventual, behavioural, and maximal authorities, but not the current ones.

LEMMA 6. *For all programs* P*, behavioural description* B*, states* σ*, and stack prefixes* ψs*:*

- $CA(\sigma, o) = CA(\psi s \cdot \sigma)$
- $EA(P, \sigma, o) \subseteq EA(P, \psi s \cdot \sigma, o)$
- $BA(B, \sigma, o) \subseteq BA(B, \psi s \cdot \sigma, o)$
- $MA(\sigma, o) \subseteq MA(\psi s \cdot \sigma, o)$

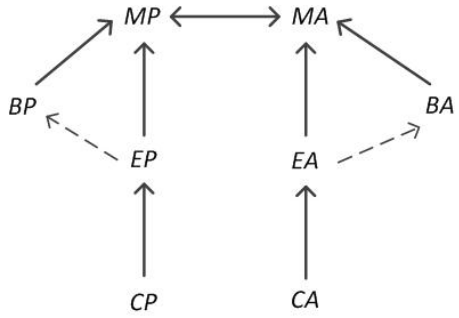

Figure 4: Relationships between Permissions and Authorities; arrows indicate subsets – dotted arrows indicate conditional subsets

5. PUTTING AUTHORITY AND PERMISSION TOGETHER

We now discuss the relation between authority and permission. Counter to initial expectation, authority does not imply permission, nor does permission imply authority. We can see this in the example shown in figure 5.

We see that `x` has authority but no permission over `y` (because `x` can invoke a method on `b` that changes `y`, but `x` has no direct reference to `y`) and that `z` has permission for `x` but no authority over it (because `z` has a direct reference to `x` but cannot cause any change to `x`). This is counter to Miller [11, ch.8] where it is claimed that eventual, behavioural, and maximal permissions are always subsets of the corresponding authorities. This is graphically depicted in figure 6

On the other hand, in the maximal world, authority and permission do indeed conflate:

LEMMA 7. *For all states* σ *and objects* o

- $MA(\sigma, o) = MP(\sigma, o)$

The proof follows from lemma 9. Figure 4 summarises these relationships.

6. CONNECTIVITY

We can observe the maximal authorities and permissions without executing the hypothetical program. Instead, we observe the transitive, symmetric closure of accessibility through the stack frames as follows:

```
class X {
  field b // initialised by new X(b)
  method thumpX { return true } //does
    nothing
}

class B {
  field y // initialised by new B(y)
  method thumpY  { y.thumpY; return false
    }
}

class Y {     //mutable object
    field  t // initialised by new Y(t)
    method thumpY { if (t) then {this.t:=
    false}
                            else {this.t:=
    true}
}                    return t }

class Z {
    field x // initialised by new Z(x)
    method thumpX { x.thumpX; return false
     }
}

var y := new Y(false)
var b := new B(y)
var x := new X(b)
var z := new Z(x)

// x points to b points to y, and x can
    mutate y
// z points to x, but z cannot mutate x
```

Figure 5: Example

The relation $\rightsquigarrow_{\phi \cdot \chi}$ represents an upper bound of the connectivity that may exist after execution of *any* method which corresponds to the frame ϕ.

DEFINITION 8. *Given a frame* ϕ *and a heap* χ *we define the relation* $\rightsquigarrow_{\phi \cdot \chi} \subseteq Address \times Address$ *as the smallest relation such that*

- $\phi(x) \rightsquigarrow_{\phi \cdot \chi} \phi(x')$ *for all* $x, x' \in dom(\phi)$
- $o \rightsquigarrow_{\phi \cdot \chi} o'$ *and* $\chi(o', f) = o''$ *for some* f*, implies* $o \rightsquigarrow_{\phi \cdot \chi} o'$
- $\rightsquigarrow_{\phi \cdot \chi}$ *is an equivalence relation*

For example, consider a frame, $\phi_1 = (\texttt{this} \mapsto 2, \texttt{x} \mapsto 5)$ and another frame, $\phi_2 = (\texttt{this} \mapsto 1)$ and heap χ_1 with objects at addresses 1, 2, 3, 4 and 5, such that $\chi_1(1, f) = 2$, $\chi_1(2, f'') = 4$, $\chi_1(1, f') = 3$ – the remaining objects have no fields, or their fields are set to `null`. The corresponding heap and the two frames are shown diagrammaitcally in figure 7.

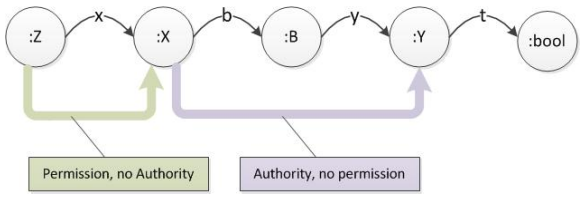

Figure 6: Authority does not imply permission, nor does permission imply authority

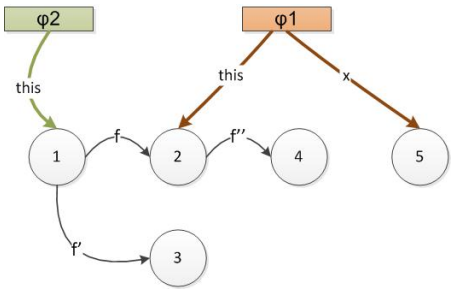

Figure 7: Example heap, and two frames.

Now consider what may happen if we execute a method with receiver and argument as in ϕ_1. We may create a link from 2 to 5, for example, by executing the statement $\texttt{this}.f := x$. In general, execution of a method on the frame ϕ_1 may create a new link between any of the objects 2, 4 and 5. By application of the definition we obtain

$\backsim_{\phi_1 \cdot \chi_1}$ =
$\{ (2,2), (2,4), (2,5), (4,2), (4,4), (4,5), (5,2), (5,4), (5,5), \}$.
Notice, that even though 5 is not connected with 4 in χ_1, the pair $(4,5)$ appears in the relationship, as 5 is reachable from the frame, and 4 is reachable from 2, which again is reachable from the frame. On the other hand, the object 1 does not appear in any pair of the relationship, as it is not reachable from the frame ϕ_1.

Similarly, by application of the definition we obtain:

$\backsim_{\phi_2 \cdot \chi_1}$ =
$\{ (1,1), (1,2), (1,3), (1,4), (2,1), (2,2), (2,3), (2,4),$
$(3,1), (3,2), (3,3), (3,4), (4,1), (4,2), (4,3), (4,4) \}$
Notice that 5 does not appear in the relationship, as it is not reachable from the frame.

We observe that the relationship $\backsim_{\phi \cdot \chi}$ represents a fully connected graph.

LEMMA 8. *For all frames ϕ, heaps χ, and objects o, o', o'', o''':*

- *$o \backsim_{\phi \cdot \chi} o'$ and $o'' \backsim_{\phi \cdot \chi} o'''$ implies $o \backsim_{\phi \cdot \chi} o'''$*

The proof is based on the fact that the relation $o \backsim_{\phi \cdot \chi} o'$ implies that the objects o and o' are reachable from the frame ϕ. This can be proven by induction on the rules from definition 8.

More importantly, we also obtain that $\backsim_\sigma$ characterises MA and MP.

LEMMA 9. *For any state σ*

- *$o \in MA(\sigma, o')$ if and only if $o \backsim_\sigma o'$*
- *$o \in MP(\sigma, o')$ if and only if $o \backsim_\sigma o'$*

To prove this lemma, we assume a "maximal" program, where each class has a method returning for each of its fields, returning the value of this field, and another method setting the value of this field.

A corollary of this lemma is that MA and MP represent an equivalence relationship, and fully connected graphs.

The relation $\backsim_{\phi \cdot \chi}$ gives an upper bound on permissions and authority. This reflects the assertion that connectivity begets connectivity [11].

We now expand our definition of maximal connectivity to talk about the complete state σ.

DEFINITION 9. *Given a state σ we define the relation $\backsim_\sigma \subseteq Address \times Address$ as the smallest relation such that*

- *$\backsim_{\phi \cdot \chi} \subseteq \backsim_\sigma$ if σ has the form $\psi s \cdot \phi \cdot \chi$.*
- *$\backsim_{\phi \cdot \chi} \subseteq \backsim_\sigma$ if σ has the form $\psi s \cdot (\phi \cdot C[\circ]) \cdot \psi s' \cdot \phi' \cdot \chi$.*
- *$\backsim_\sigma$ is an equivalence relation*

For example, $3 \backsim_{\phi_1 \cdot \ldots \cdot \phi_2 \cdot \chi_1} 5$.

Increasing the stack prefix increases the relation $\backsim$:

LEMMA 10. *For any state σ, stack prefix ψs, and objects o and o'*

- *$\backsim_\sigma \subseteq \backsim_{\psi s \cdot \sigma}$*

7. CONCLUSION AND DISCUSSION

In this short paper we have revisited the object-capability concepts of permission and authority, proposed precise foundational definitions, and clarified the relationships between the definitions.

The distinction between permissions and authority, and their various flavours is essential for the sound analysis of threats to open systems. The aim of vulnerability analysis is to calculate precisely the eventual authority (EA), however this is often too difficult, and for this we want to use safe upper bounds. Thus, in case of conformance, BA is a good and often tractable approximation, and has been used in [2, 5].

Authority of various kinds has been used to verify or analyse object-capability systems [10, 2]. In contrast, the contribution of this short paper is to provide crisp definitions of permission, authority, and their future bounds, and the relations between them.

We consider that the definitions given in this paper constitute a first step towards a crystallisation of the meaning of the concepts of permission and authority. However, these definitions need to be expanded in several ways:

First, the concept of adherence to specification ($P \subset B$) is very strict, as it requires the executions in P and those in B to have the same granularity. A more general approach should allow them to run at different granularity but would require them to "agree" at some future point in execution, e.g. $P \vdash \sigma, \texttt{stmts} \rightsquigarrow^* \sigma', \texttt{sr}$ implies that there exists a σ'' such that $P \vdash \sigma', \texttt{stmts} \rightsquigarrow^* \sigma'', \texttt{sr}$ and $B \vdash \sigma, \texttt{stmts} \overset{?}{\rightsquigarrow}{}^* \sigma'', \texttt{sr}$. However in such a case the third inclusion property in lemma 1 will not hold. We will investigate how to adapt the definition so as to obtain some weaker version of this inclusion property.

Second, the definitions of possible worlds caters for the situation where the program is either known (EW) or is unknown (MW). However, it may be useful to refine the model to allow the current known program P to be linked against unknown programs P', and thus argue that P is robust against the unknown code P' (the adversary). In further work we will investigate refinements of the future worlds definitions to reflect this distinction.

Third, our definition of authority, **Definition 6**, models authority as a *sufficient cause* for an effect: an object o has authority over an object o' if it can call methods which will modify the state of o'. An alternative view models authority as a *necessary cause* for an effect: an object o has authority over an object o' if o' cannot be modified except via object o [12]. We will investigate this approach in further work.

Fourth, we plan to investigate how static or dynamic type information (e.g. ownership) should allow us to make more precise bounds, which will be reflected in B.

Finally, and most importantly, we plan to use these definitions to support the verification of a wide-range of properties of object-capability systems [4].

Acknowledgments.

We thank the anonymous referees for their comments. This work is partially supported by a James Cook Fellowship and Royal Society of New Zealand Marsden Fund, and by the EU FP7 project Upscale.

References

[1] D. G. Clarke, J. M. Potter, and J. Noble. Ownership types for flexible alias protection. In *OOPSLA*. ACM, 1998.

[2] D. Devriese, L. Birkedal, and F. Piessens. Reasoning about object capabilities with logical relations and effect parametricity. In *Euro S & P*, March 2016.

[3] S. Drossopoulou and J. Noble. The need for capability policies. In *FTfJP*, 2013.

[4] S. Drossopoulou and J. Noble. How to break the bank: Semantics of capability policies. In *iFM*, 2014.

[5] S. Drossopoulou, J. Noble, and M. S. Miller. Swapsies on the Internet. In *PLAS*, 2015.

[6] L. Jia, S. Sen, D. Garg, and A. Datta. A logic of programs with interface-confined code. In *CSF*, pages 512–525, 2015.

[7] B. W. Lampson. Protection. *Operating Systems Review*, 8(1):18–24, Jan. 1974.

[8] G. T. Leavens, E. Poll, C. Clifton, Y. Cheon, C. Ruby, D. R. Cok, P. Müller, J. Kiniry, and P. Chalin. JML Reference Manual. Iowa State Univ. `www.jmlspecs.org`, February 2007.

[9] K. R. M. Leino and P. Müller. Object invariants in dynamic contexts. In *ECOOP*, Springer, 2004.

[10] S. Maffeis, J. Mitchell, and A. Taly. Object capabilities and isolation of untrusted web applications. In *IEEE S & P*, 2010.

[11] M. S. Miller. *Robust Composition: Towards a Unified Approach to Access Control and Concurrency Control*. PhD thesis, Baltimore, Maryland, 2006.

[12] T. Murray. *Analysing the Security Properties of Object-Capability Patterns*. D.Phil. thesis, University of Oxford, 2010.