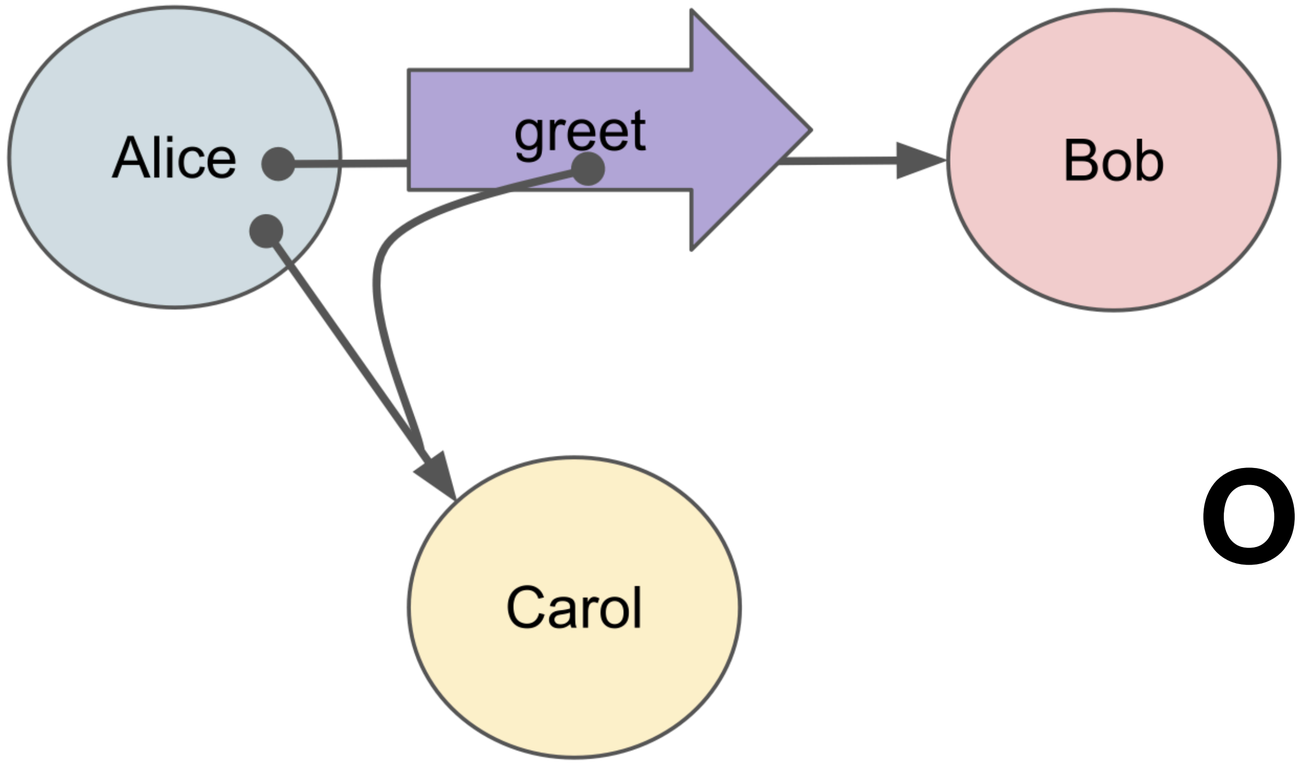


Reasoning with Object Capabilities

James Noble

Susan Eisenbach

Julian Mackay

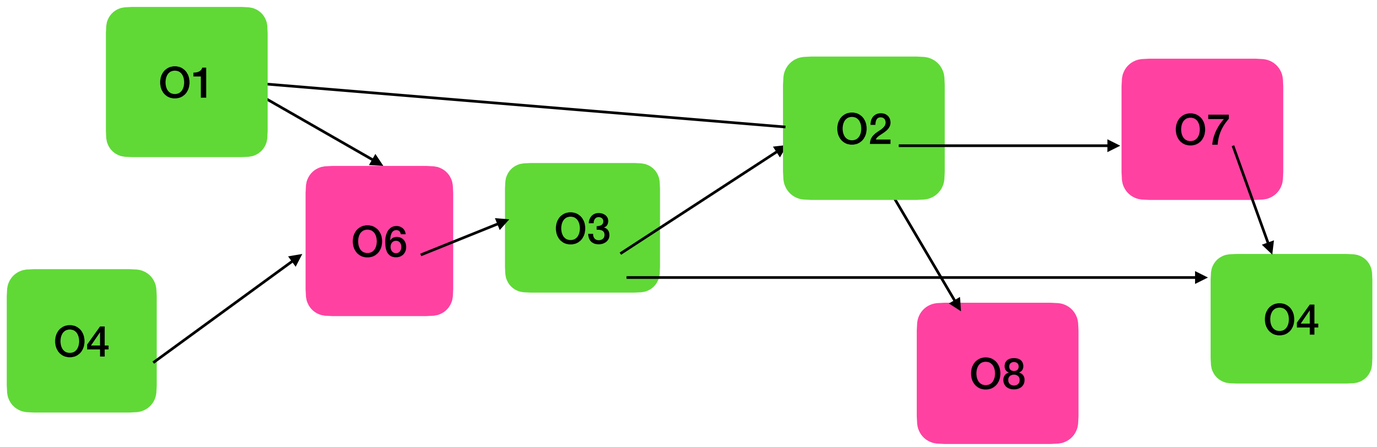
and in earlier works

Mark Miller

Toby Murray

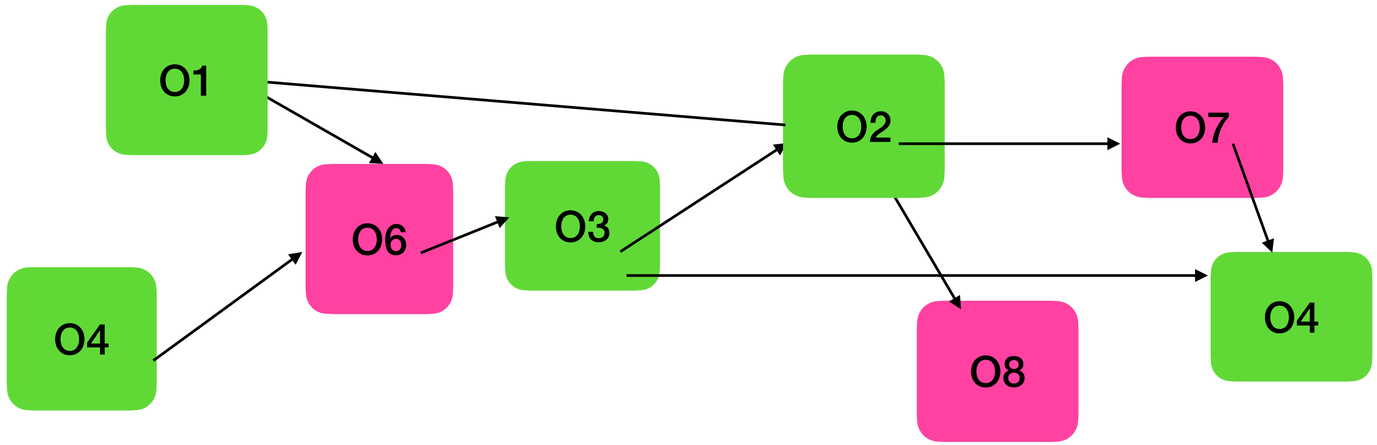
Our research Question

Reason about how our, internal, trusted objects can interact securely with the
external, untrusted, potentially malicious, objects.



Our research Question

Reason about how our, internal, trusted objects can interact securely with the
external, untrusted, potentially malicious, objects. using capabilities.



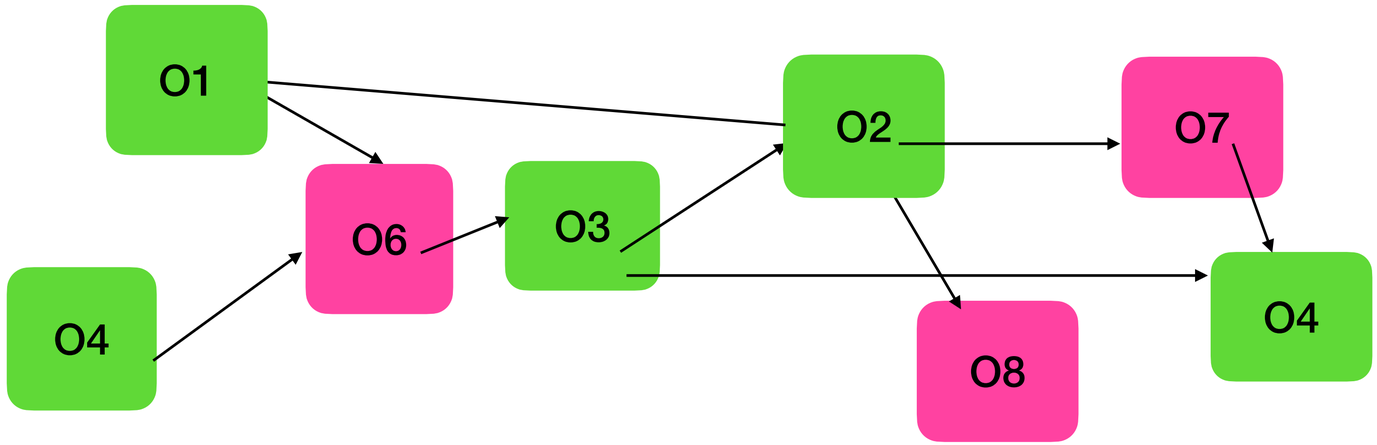
Our research Question

Reason about how our, internal, trusted objects can interact securely with the
external, untrusted, potentially malicious, objects. using capabilities.

Literature:

*“A **capability** describes a transferable right to perform one (or more) operations.”*

“References cannot be forged.
Capability transferred only through messages or through creation.”



“A capability describes a transferable right to perform one (or more) operations.”

Literature:

“References cannot be forged. Capability transferred only through messages or through creation.”

“A capability describes a transferable right to perform one (or more) operations.”

Literature:

“References cannot be forged. Capability transferred only through messages or through creation.”

Our Insights:

Capabilities are necessary conditions for *effects* (not operations)

***Tracking access* to capabilities is paramount.**

Example
and
Four Challenges

Three Modules

.. assuming all methods are public, and fields are private

```
module Mod_good
  class Account
    field balance:int
    field pwd: Password
    method transfer(dest:Account, pwd':Password) -> void
      if this.pwd==pwd'
        this.balance-=100
        dest.balance+=100
     method init(pwd':Password) -> void
      if this.pwd==null
        this.pwd=pwd'
  class Password
```

Three Modules

.. assuming all methods are public, and fields are private

```
module Mod_good
  class Account
    field balance:int
    field pwd: Password
    method transfer(dest:Account, pwd':Password) -> void
      if this.pwd==pwd'
        this.balance-=100
        dest.balance+=100
     method init(pwd':Password) -> void
      if this.pwd==null
        this.pwd=pwd'
  class Password
```

```
module Mod_bad
  class Account
    field balance:int
    field pwd: Password
    method transfer(..) ...
      ... as earlier ...
    method init(...) ...
       ... as earlier ...
    method set(pwd': Password)
      this.pwd=pwd'

  class Password
```

Three Modules

.. assuming all methods are public, and fields are private

```
module Mod_good
  class Account
    field balance:int
    field pwd: Password
    method transfer(dest:Account, pwd':Password) -> void
      if this.pwd==pwd'
        this.balance-=100
        dest.balance+=100
     method init(pwd':Password) -> void
      if this.pwd==null
        this.pwd=pwd'
  class Password
```

```
module Mod_bad
  class Account
    field balance:int
    field pwd: Password
    method transfer(..) ...
      ... as earlier ...
    method init(...) ...
       ... as earlier ...
    method set(pwd': Password)
      this.pwd=pwd'

  class Password
```

Three Modules

.. assuming all methods are public, and fields are private

```
module Mod_good
  class Account
    field balance:int
    field pwd: Password
    method transfer(dest:Account, pwd':Password) -> void
      if this.pwd==pwd'
        this.balance-=100
        dest.balance+=100
     method init(pwd':Password) -> void
      if this.pwd==null
        this.pwd=pwd'
  class Password
```

```
module Mod_bad
  class Account
    field balance:int
    field pwd: Password
    method transfer(..) ...
      ... as earlier ...
    method init(...) ...
       ... as earlier ...
    method set(pwd': Password)
      this.pwd=pwd'

  class Password
```

```
module Mod_better
  class Account
    field balance:int
    field pwd: Password
    method transfer(..)
      ... as earlier ...

    method set(pwd',pwd'': Password)
      if (this.pwd==pwd')
        this.pwd=pwd''
  class Password
```

Three Modules

.. assuming all methods are public, and fields are private

```
module Mod_good
  class Account
    field balance:int
    field pwd: Password
    method transfer(dest:Account, pwd':Password) -> void
      if this.pwd==pwd'
        this.balance-=100
        dest.balance+=100
     method init(pwd':Password) -> void
      if this.pwd==null
        this.pwd=pwd'
  class Password
```

```
module Mod_bad
  class Account
    field balance:int
    field pwd: Password
    method transfer(..) ...
      ... as earlier ...
    method init(...) ...
       ... as earlier ...
    method set(pwd': Password)
      this.pwd=pwd'

  class Password
```

```
module Mod_better
  class Account
    field balance:int
    field pwd: Password
    method transfer(..)
      ... as earlier ...

    method set(pwd',pwd'': Password)
      if (this.pwd==pwd')
        this.pwd=pwd''
  class Password
```

```
module Mod_good
  class Account
    field balance:int
    field pwd: Password
    method transfer(dest:Account, pwd':Password) ->
      if this.pwd==pwd'
        this.balance-=100
        dest.balance+=100
     method init(pwd':Password) -> void
      if this.pwd==null
        this.pwd=pwd'
  class Password
```

Challenge_1: A module spec S, such that

$M_{good} \models S$

$M_{bad} \not\models S$

$M_{better} \models S$

```
module Mod_bad
  class Account
    field balance:int
    field pwd: Password
    method transfer(..) ...
      ... as earlier ...
    method init(...) ...
       ... as earlier ...
    method set(pwd': Password)
      this.pwd=pwd'

  class Password
```

```
module Mod_better
  class Account
    field balance:int
    field pwd: Password
    method transfer(..)
      ... as earlier ...

    method set(pwd',pwd'': Password)
      if (this.pwd==pwd')
        this.pwd=pwd''
  class Password
```

```
module Mod_good
  class Account
    field balance:int
    field pwd: Password
    method transfer(dest:Account, pwd':Password) ->
      if this.pwd==pwd'
        this.balance-=100
        dest.balance+=100
     method init(pwd':Password) -> void
      if this.pwd==null
        this.pwd=pwd'
  class Password
```

Challenge_2: An inference system, such that

$M_{good} \vdash S$

$M_{bad} \nvdash S$

$M_{better} \vdash S$

```
module Mod_bad
  class Account
    field balance:int
    field pwd: Password
    method transfer(..) ...
      ... as earlier ...
    method init(...) ...
      ... as earlier ...
    method set(pwd': Password)
      this.pwd=pwd'

  class Password
```

```
module Mod_better
  class Account
    field balance:int
    field pwd: Password
    method transfer(..)
      ... as earlier ...

    method set(pwd',pwd'': Password)
      if (this.pwd==pwd')
        this.pwd=pwd''
  class Password
```

```
module Mod_good
  class Account
    field balance:int
    field pwd: Password
    method transfer(dest:Account, pwd':Password) ->
      if this.pwd==pwd'
        this.balance-=100
        dest.balance+=100
     method init(pwd':Password) -> void
      if this.pwd==null
        this.pwd=pwd'
  class Password
```

Challenge_3: Inference system should be algorithmic

```
module Mod_bad
  class Account
    field balance:int
    field pwd: Password
    method transfer(..) ...
      ... as earlier ...
    method init(...) ...
       ... as earlier ...
    method set(pwd': Password)
      this.pwd=pwd'

  class Password
```

```
module Mod_better
  class Account
    field balance:int
    field pwd: Password
    method transfer(..)
      ... as earlier ...

    method set(pwd',pwd'': Password)
      if (this.pwd==pwd')
        this.pwd=pwd''
  class Password
```

```
1  module Mod1
2        ...
3      method cautious(untrusted:Object)
4          a = new Account
5          p = new Password
6          a.set(null,p)
7          ...
8          untrusted.make_payment(a)
9          ...
```

```
1  module Mod1
2        ...
3      method cautious(untrusted:Object)
4          a = new Account
5          p = new Password
6          a.set(null,p)
7          ...
8          untrusted.make_payment(a)
9          ...
```

```
1  module Mod1
2        ...
3      method cautious(untrusted:Object)
4          a = new Account
5          p = new Password
6          a.set(null,p)
7          ...
8          untrusted.make_payment(a)
9          ...
```

If `Account` comes from a “good” module,
and on line 3 `untrusted` has no access to `a.password`,
and line 7 does not leak ta`.password` to `untrusted`,

```
1  module Mod1
2        ...
3      method cautious(untrusted:Object)
4          a = new Account
5          p = new Password
6          a.set(null,p)
7          ...
8          untrusted.make_payment(a)
9          ...
```

If `Account` comes from a “good” module,
and on line 3 `untrusted` has no access to `a.password`,
and line 7 does not leak t`a.password` to `untrusted`,
then
the balance of `a` does not decrease by the call `make_payment`
(line 8).

```
1  module Mod1
2        ...
3      method cautious(untrusted:Object)
4          a = new Account
5          p = new Password
6          a.set(null,p)
7          ...
8          untrusted.make_payment(a)
9          ...
```

External call

Challenge_4: An inference system, such that we can prove **external** calls.

If `Account` comes from a “good” module,
and on line 3 `untrusted` has no access to `a.password`,
and line 7 does not leak `a.password` to `untrusted`,
then
the balance of `a` does not decrease by the call `make_payment`
(line 8).

Challenge_1: A module spec S, such that

$M_{good} \models S$

$M_{bad} \not\models S$

$M_{better} \models S$

Challenge_1: A module spec S, such that

$$M_{good} \models S$$
$$M_{bad} \not\models S$$
$$M_{better} \models S$$

Remember: A capability represents a transferable right
to perform one or more operations on a given object

Challenge_1: A module spec S, such that

$M_{good} \models S$
$M_{bad} \not\models S$
$M_{better} \models S$

Remember: A capability represents a transferable right
to perform one or more operations on a given object

So: “The password enables withdrawal from the account”?

Challenge_1: A module spec S, such that

$M_{good} \models S$
$M_{bad} \not\models S$
$M_{better} \models S$

Remember: A capability represents a transferable right
to perform one or more operations on a given object

So: “The password enables withdrawal from the account”?

Or: “Without the password call of withdraw will fail”?

Challenge_1: A module spec S, such that

$M_{good} \models S$
$M_{bad} \not\models S$
$M_{better} \models S$

Remember: A capability represents a transferable right
to perform one or more operations on a given object

~~***So***: "The password enables withdrawal from the account"?~~

Or: "Without the password call of withdraw will fail"?

Challenge_1: A module spec S, such that

$M_{good} \models S$
$M_{bad} \not\models S$
$M_{better} \models S$

Remember: A capability represents a transferable right
to perform one or more operations on a given object

~~***So***: "The password enables withdrawal from the account"?~~

Or: "Without the password call of withdraw will fail"?

Or: "Without the password no reduction of the balance of the account"?

Challenge_1: A module spec S, such that

$M_{good} \models S$
$M_{bad} \not\models S$
$M_{better} \models S$

Remember: A capability represents a transferable right
to perform one or more operations on a given object

~~***So***: "The password enables withdrawal from the account"?~~

~~***Or***: "Without the password call of withdraw will fail"?~~

Or: "Without the password no reduction of the balance of the account"?

Challenge_1: A module spec S, such that

$M_{good} \models S$
$M_{bad} \not\models S$
$M_{better} \models S$

Remember: A capability represents a transferable right
to perform one or more operations on a given object

~~***So***: "The password enables withdrawal from the account"?~~

~~***Or***: "Without the password call of withdraw will fail"?~~

Or: "Without the password no reduction of the balance of the account"?

So: $\forall$ s:Statement. {without a.password $\wedge$ a.balance=b} s { a.balance >= b }

Challenge_1: A module spec S, such that

$M_{good} \models S$
$M_{bad} \not\models S$
$M_{better} \models S$

Motto:
Capability is a *necessary* condition for some effect

Remember: A capability represents a transferable right to perform one or more operations on a given object

~~***So***: "The password enables withdrawal from the account"?~~

~~***Or***: "Without the password call of withdraw will fail"?~~

Or: "Without the password no reduction of the balance of the account"?

So: $\forall$ s:Statement. {without a.password $\wedge$ a.balance=b} s { a.balance >= b }

Challenge_1: A module spec S, such that …

So: $\forall$ s:Statement {without a.password $\wedge$ a.balance=b} s { a.balance >= b }

Challenge_1: A module spec S, such that …

So: ∀ s:Statement {without a.password ∧ a.balance=b} s { a.balance >= b }

So: ∀ a: Account. ∀ n: Num. (| a.password **prt** ∧ a.balance=b |) (| a.balance>=b |)

Challenge_1: A module spec S, such that …

So: ∀ s:Statement {without a.password ∧ a.balance=b} s { a.balance >= b }

So: ∀ a: Account. ∀ n: Num. (| a.password **prt** ∧ a.balance=b |) (| a.balance>=b |)

In general: ∀ x1:C1,x2:C2… (| A |) (| A' |)

Challenge_1: A module spec S, such that …

So: ∀ s:Statement {without a.password ∧ a.balance=b} s { a.balance >= b }

So: ∀ a: Account. ∀ n: Num. (| a.password **prt** ∧ a.balance=b |) (| a.balance>=b |)

In general: ∀ x1:C1,x2:C2… (| A |) (| A' |)

Challenge_1_a : Meaning of x **prt**

Challenge_1: A module spec S, such that …

So: ∀ s:Statement {without a.password ∧ a.balance=b} s { a.balance >= b }

So: ∀ a: Account. ∀ n: Num. (| a.password **prt** ∧ a.balance=b |) (| a.balance>=b |)

In general: ∀ x1:C1,x2:C2… (| A |) (| A' |)

Challenge_1_a : Meaning of x **prt**

Challenge_1_b : Meaning of ∀ x1:C1,x2:C2… (| A |) (| A' |)

Challenge_1_a : Meaning of x **prt**

Remember:

…how our, internal, trusted objects will interact securely with the external, untrusted, potentially malicious, objects.

Def: o **prt-frm** o’, if any path from o’ to o goes through an internal object.

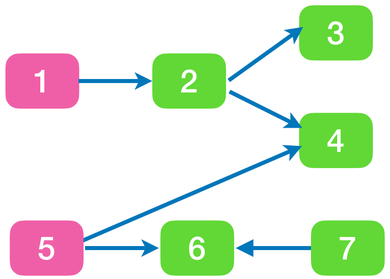
Challenge_1_a : Meaning of x **prt**

Remember:

…how our, internal, trusted objects will interact securely with the external, untrusted, potentially malicious, objects.

Def: o **prt-frm** o’, if any path from o’ to o goes through an internal object.

For example:



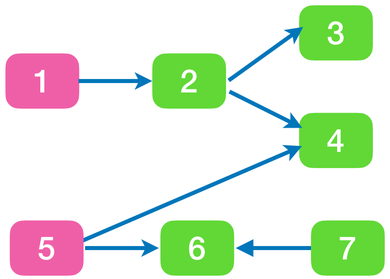
Challenge_1_a : Meaning of x **prt**

Remember:

…how our, internal, trusted objects will interact securely with the external, untrusted, potentially malicious, objects.

Def: o **prt-frm** o', if any path from o' to o goes through an internal object.

For example:



o3 **prt-frm** o1

o4 **prt-frm** o1

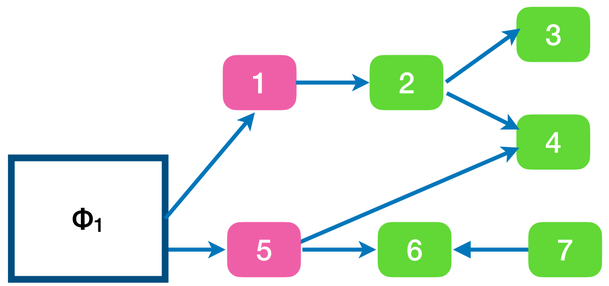
¬ (o4 **prt-frm** o5)

Challenge_1_a : Meaning of. x **prt**

Def: o **prt-frm o'**, if **extr** o' and any path from o' to o goes through an internal object.

Def: o **prt** , if o **prt-from** any external object accessible from top frame

For example:

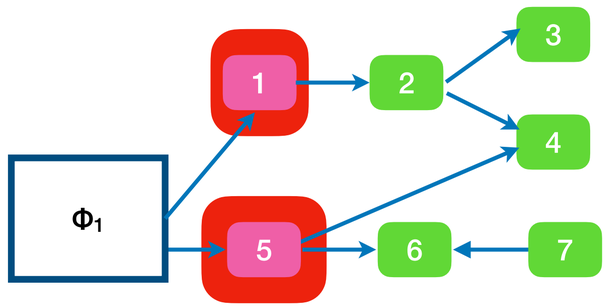


Challenge_1_a : Meaning of. x **prt**

Def: o **prt-frm o'**, if **extr** o' and any path from o' to o goes through an internal object.

Def: o **prt** , if o **prt-from** any external object accessible from top frame

For example:

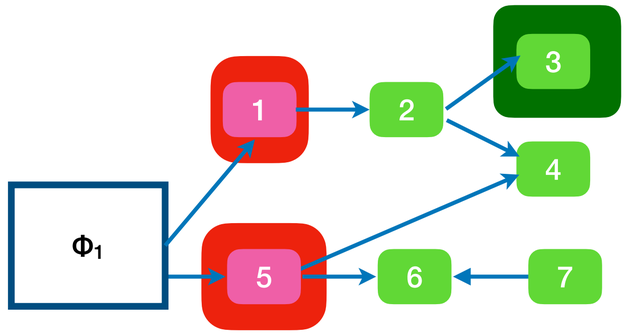


Challenge_1_a : Meaning of. x **prt**

Def: o **prt-frm o'**, if **extr** o' and any path from o' to o goes through an internal object.

Def: o **prt** , if o **prt-from** any external object accessible from top frame

For example:

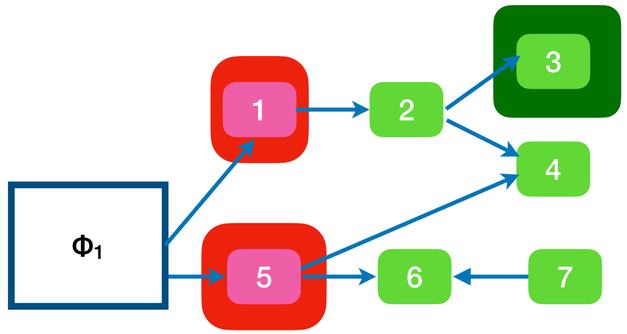


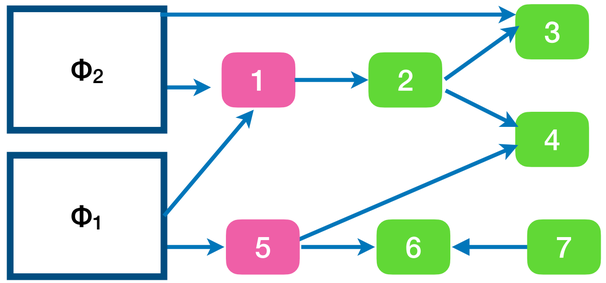
Challenge_1_a : Meaning of. x **prt**

Def: o **prt-frm o'**, if **extr** o' and any path from o' to o goes through an internal object.

Def: o **prt** , if o **prt-from** any external object accessible from top frame

For example:



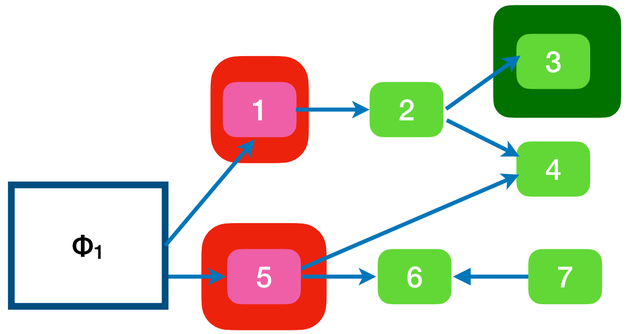


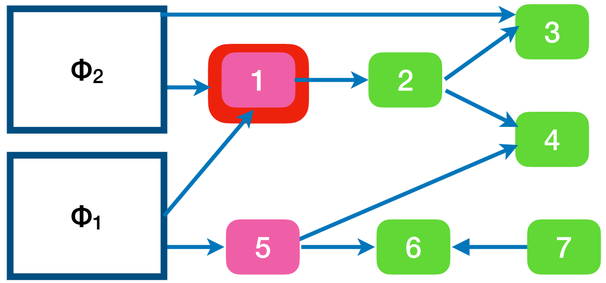
Challenge_1_a : Meaning of. x **prt**

Def: o **prt-frm o'**, if **extr** o' and any path from o' to o goes through an internal object.

Def: o **prt** , if o **prt-from** any external object accessible from top frame

For example:



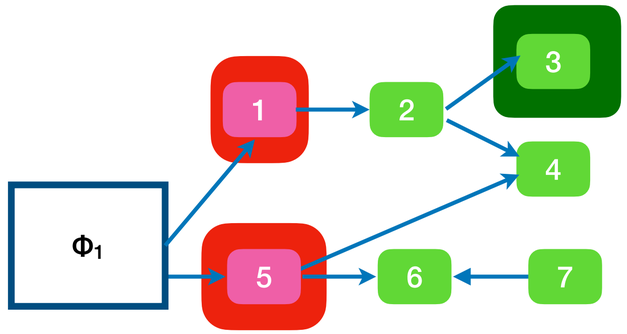


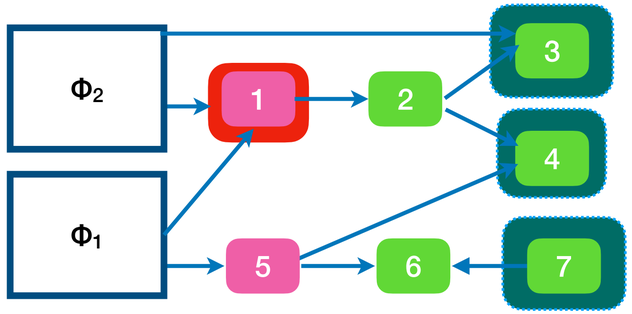
Challenge_1_a : Meaning of. x **prt**

Def: o **prt-frm o'**, if **extr** o' and any path from o' to o goes through an internal object.

Def: o **prt** , if o **prt-from** any external object accessible from top frame

For example:





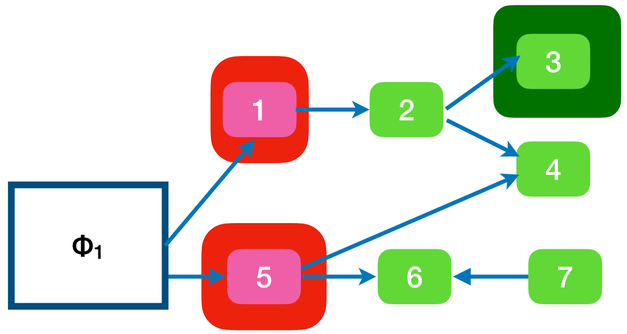
Challenge_1_a : Meaning of. x **prt**

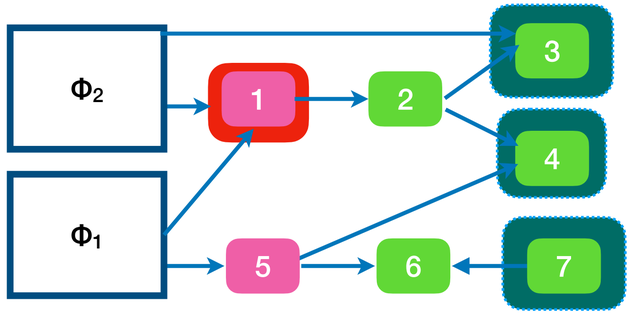
Def: o **prt-frm o'**, if **extr** o' and any path from o' to o goes through an internal object.

Def: o **prt** , if o **prt-from** any external object accessible from top frame

Motto:
Tracking access to Capabilities

For example:





Challenge_1_b : Meaning of ∀ x1:C1,x2:C2… (| A |) (| A’ |)

Challenge_1_b : Meaning of ∀ x1:C1,x2:C2… (| A |) (| A’ |)

For all states σ arising from execution of our internal module, combined with any possible external module,

Challenge_1_b : Meaning of ∀ x1:C1,x2:C2… (| A |) (| A’ |)

For all states σ arising from execution of our internal module, combined with any possible external module,

for all future states σ’ arising from σ whithout returning from top frame,

Challenge_1_b : Meaning of ∀ x1:C1,x2:C2… (| A |) (| A’ |)

For all states σ arising from execution of our internal module, combined with any possible external module,

for all future states σ’ arising from σ whithout returning from top frame,

for all globally accessible objects x1, .. xn of class C1, … Cn,

Challenge_1_b : Meaning of ∀ x1:C1,x2:C2… (| A |) (| A’ |)

For all states σ arising from execution of our internal module, combined with any possible external module,

for all future states σ’ arising from σ whithout returning from top frame,

for all globally accessible objects x1, .. xn of class C1, … Cn,

If σ satisfies A

Challenge_1_b : Meaning of ∀ x1:C1,x2:C2… (| A |) (| A’ |)

For all states σ arising from execution of our internal module, combined with any possible external module,

for all future states σ’ arising from σ whithout returning from top frame,

for all globally accessible objects x1, .. xn of class C1, … Cn,

If σ satisfies A

Then

Challenge_1_b : Meaning of ∀ x1:C1,x2:C2… (| A |) (| A’ |)

For all states σ arising from execution of our internal module, combined with any possible external module,

for all future states σ’ arising from σ whithout returning from top frame,

for all globally accessible objects x1, .. xn of class C1, … Cn,

If σ satisfies A

Then

σ' satisfies A'

Putting it together

Putting it together

Therefore,

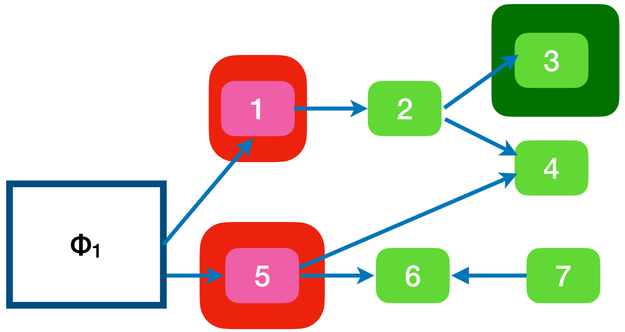
$\forall$ x: Object (| **prt** x $\wedge$ A(x) |) (| A(x) |)

Putting it together

Therefore,

$\forall$ x: Object (| **prt** x $\wedge$ A(x) |) (| A(x) |)

guarantees that if we start below,

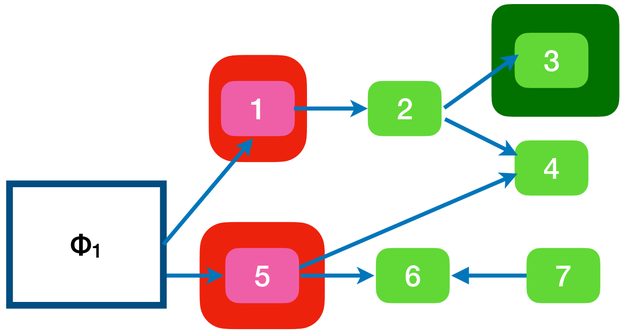


Putting it together

Therefore,

$\forall$ x: Object (| **prt** x $\wedge$ A(x) |) (| A(x) |)

guarantees that if we start below,



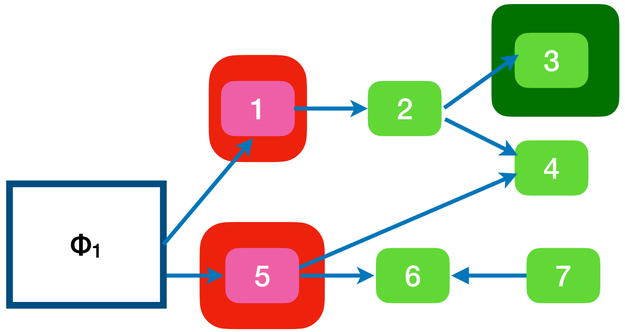
then, A(o3) will be preserved

Putting it together

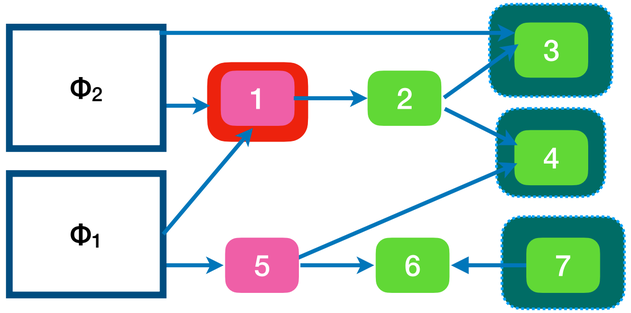
Therefore,

$\forall$ x: Object (| **prt** x $\wedge$ A(x) |) (| A(x) |)

guarantees that if we start below,



And if we start below,



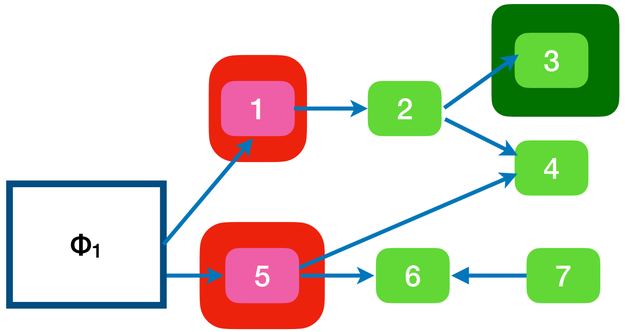
then, A(o3) will be preserved

Putting it together

Therefore,

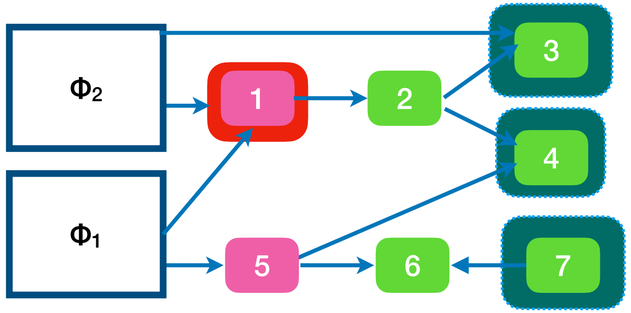
$\forall$ x: Object (| **prt** x $\wedge$ A(x) |) (| A(x) |)

guarantees that if we start below,



then, A(o3) will be preserved

And if we start below,



then, A(o3) and A(o4) and A(o7) will be preserved

Challenge_1: A module spec S, such that

$M_{good} \models S$

$M_{bad} \not\models S$

$M_{better} \models S$

$S =$

$\forall$ a:Account (| **prt** a |) (| **prt** a |)

$\wedge$

$\forall$ a:Account (| **prt** a.password |) (| **prt** a.password |)

$\wedge$

$\forall$ a:Account. $\forall$ b:Num (| **prt** a $\wedge$ a.balance = b |) (| a.balance = b |)

$\wedge$

$\forall$ a:Account. $\forall$ b:Num (| **prt** a.password $\wedge$ a.balance = b |) (| **prt** a.balance $\geq$ b |)

Challenge_1: A module spec S, such that

$M_{good} \models S$

$M_{bad} \not\models S$

$M_{better} \models S$

API - agnostic:
a.balance, a.password can be ghost

Talks about effects

$$
\begin{aligned}
S = \; & \forall\, a{:}\mathrm{Account}\ (|\ \mathbf{prt}\ a\ |)\ (|\ \ \mathbf{prt}\ a\ |) \\
& \wedge \\
& \forall\, a{:}\mathrm{Account}\ (|\ \mathbf{prt}\ a.\mathrm{password}\ |)\ (|\ \ \mathbf{prt}\ a.\mathrm{password}\ |) \\
& \wedge \\
& \forall\, a{:}\mathrm{Account}.\ \forall\, b{:}\mathrm{Num}\ (|\ \mathbf{prt}\ a \wedge a.\mathrm{balance} = b\ |)\ (|\ \ a.\mathrm{balance} = b\ |) \\
& \wedge \\
& \forall\, a{:}\mathrm{Account}.\ \forall\, b{:}\mathrm{Num}\ (|\ \mathbf{prt}\ a.\mathrm{password} \wedge a.\mathrm{balance} = b\ |)\ (|\ \ \mathbf{prt}\ a.\mathrm{balance} \geq b\ |)
\end{aligned}
$$

Talks about emergent behaviour

Challenge_2: An inference system, such that

$M_{good} \vdash S$

$M_{bad} \nvdash S$

$M_{better} \vdash S$

Challenge_2: An inference system, such that …

An assertion A is **encapsulated** by module M, if
it can only be invalidated through calls to methods from M.

Challenge_2: An inference system, such that …

An assertion A is **encapsulated** by module M, if
it can only be invalidated through calls to methods from M.

For example:

$$\texttt{Mod}_{\texttt{bad}} \vDash \textit{Encaps}(\texttt{a}:\texttt{Account} \wedge \texttt{a.balance} = \texttt{bal})$$
$$\texttt{Mod}_{\texttt{better}} \vDash \textit{Encaps}(\texttt{a}:\texttt{Account} \wedge \texttt{a.balance} = \texttt{bal})$$

Challenge_2: An inference system, such that …

An assertion A is **encapsulated** by module M, if
it can only be invalidated through calls to methods from M.

For example:

$$\mathtt{Mod}_{\mathtt{bad}} \vDash \mathit{Encaps}(\mathtt{a:Account} \wedge \mathtt{a.balance} = \mathtt{bal})$$
$$\mathtt{Mod}_{\mathtt{better}} \vDash \mathit{Encaps}(\mathtt{a:Account} \wedge \mathtt{a.balance} = \mathtt{bal})$$

Assume two further modules, Mod_{ul} and Mod_{pl}, which use ledgers to keep a map between accounts and their balances, which export functions that allow the update of this map. In Mod_{ul} the ledger is *not* protected, while in Mod_{pl} the ledger *is* protected.

$$Mod_{ul} \nvDash \mathit{Encaps}(\mathtt{a:Account} \wedge \mathtt{a.balance} = \mathtt{bal})$$
$$Mod_{pl} \vDash \mathit{Encaps}(\mathtt{a:Account} \wedge \mathtt{a.balance} = \mathtt{bal})$$

Challenge_2: An inference system, such that …

TWO-STATE

???

$$\frac{}{M \vdash \forall \overline{x : C}.(\!| A |\!)(\!| A |\!)}$$

Challenge_2: An inference system, such that …

TWO-STATE

$$\frac{\begin{array}{c} M \vdash \mathit{Encaps}(\overline{x : C} \wedge A) \\ \textit{for all public methods from } D, \textit{ with } \texttt{mBody}(m, D, M) = \overline{y : D}\{\, s \,\} \\ M \vdash \{\overline{x : C} \wedge A \wedge \texttt{this} : \texttt{D} \wedge \overline{y : D}\}\, s\, \{\, A \,\} \end{array}}{M \vdash \forall \overline{x : C}.(\!|\, A \,|\!)(\!|\, A \,|\!)}$$

Challenge_2: An inference system, such that …

TWO-STATE

$$\frac{\begin{array}{c} M \vdash \mathit{Encaps}(\overline{x : C} \wedge A) \\ \text{for all public methods from } D, \text{ with } \mathtt{mBody}(m, D, M) = \overline{y : D}\{\, s \,\} \\ M \vdash \{\overline{x : C} \wedge A \wedge \mathtt{this} : \mathtt{D} \wedge \overline{y : D}\}\, s\, \{\, A \,\} \end{array}}{M \vdash \forall \overline{x : C}.(\!|A|\!)(\!|A|\!)}$$

WEAKEN

$$\frac{M \vdash S \qquad M \vdash S \sqsubseteq S'}{M \vdash S'}$$

MULTI

$$\frac{M \vdash S \qquad M \vdash S'}{M \vdash S \wedge S'}$$

Challenge_4: An inference system, such that we can prove external calls

Challenge_4: An inference system, such we can prove external calls

ExtCall

$$M \vdash A \rightarrow (\ \mathbf{extl}\, z \ \wedge\ \mathbf{extl}\, \overline{u} \ \wedge\ \underline{\mathbf{intl}\, \overline{y}}\ ) \qquad \overline{v} \text{ is a permutation of } \overline{u}, \overline{y}$$

$$\frac{}{M \vdash \{\ A\ \}\ z.m(\overline{v})\ \{\ \quad ???\quad \ \}}$$

Challenge_4: An inference system, such we can prove external calls

ExtCall

$$M \vdash A \rightarrow (\ \mathbf{extl}\, z \ \wedge\ \mathbf{extl}\, \overline{u} \ \wedge\ \underline{\mathbf{intl}\, \overline{y}}\ ) \qquad \overline{v} \text{ is a permutation of } \overline{u}, \overline{y}$$

$$\frac{}{M \vdash \{\ A\ \}\ z.m(\overline{v})\ \{\qquad ???\qquad\ \}}$$

Motto:
Capability is a *necessary* condition
for some effect

Challenge_4: An inference system, such we can prove external calls

ExtCall

$$\frac{\begin{array}{l} M \vdash A \rightarrow (\ \mathbf{extl}\, z \ \wedge\ \mathbf{extl}\, \overline{u} \ \wedge\ \mathbf{intl}\, \overline{y}\ ) \qquad \overline{v} \text{ is a permutation of } \overline{u}, \overline{y} \\ \vdash\ M : \forall \overline{x : C}.(\!|A_1|\!)(\!|A_2|\!) \\ M \vdash\ A \rightarrow (\overline{x : C} \wedge A_1) \end{array}}{M \vdash \{\ A\ \}\, z.m(\overline{v})\, \{\ A_2\ \}}$$

Challenge_4: An inference system, such we can prove external calls

ExtCall

$$\frac{\begin{array}{c} M \vdash A \;\rightarrow\; (\; \mathbf{extl}\, z \;\wedge\; \mathbf{extl}\, \overline{u} \;\wedge\; \mathbf{intl}\, \overline{y} \;) \qquad \overline{v} \text{ is a permutation of } \overline{u}, \overline{y} \\ \vdash\; M \;:\; \forall \overline{x : C}.(\!|\, A_1 \,|\!)(\!|\, A_2 \,|\!) \\ M \vdash Lift(A, \{z, \overline{u}\}, \overline{y}) \;\rightarrow\; (\; \overline{x : C} \wedge A_1 \;) \end{array}}{M \;\vdash\; \{\; A \;\}\; z.m(\overline{v})\; \{\qquad A_2 \qquad\}}$$

$$\begin{array}{lcll} Lift(v = v', \overline{z}, \overline{y}) & = & v = v' & \\ Lift(x.f = v, \overline{z}, \overline{y}) & = & x.f = v & \\ Lift(\,\mathbf{prt}\, x\,, \overline{z}, \overline{y}) & = & \mathbf{prt}\, x & \\ Lift(\,x\, \mathbf{prt\text{–}frm}\, \overline{u}, \overline{z}, \overline{y}) & = & \mathbf{prt}\, x & \text{if } \overline{z} \subseteq \overline{u}, \text{ and } x \notin \overline{y} \\ & = & \texttt{true} & \text{otherwise} \end{array}$$

...

Challenge_4: An inference system, such we can prove external calls

ExtCall

$$\frac{\begin{array}{c} M \vdash A \;\rightarrow\; (\; \mathbf{extl}\, z \;\wedge\; \mathbf{extl}\, \overline{u} \;\wedge\; \mathbf{intl}\, \overline{y} \;) \qquad \overline{v} \text{ is a permutation of } \overline{u}, \overline{y} \\ \vdash\; M \;:\; \forall \overline{x : C}.(\!|\, A_1 \,|\!)(\!|\, A_2 \,|\!) \\ M \vdash Lift(A, \{z, \overline{u}\}, \overline{y}) \;\rightarrow\; (\, \overline{x : C} \wedge A_1 \,) \end{array}}{M \;\vdash\; \{\; A \;\}\; z.m(\overline{v})\; \{\; Lower(A_2) \qquad \}}$$

$$\begin{array}{lcl} Lower(v = v') & = & v = v' \\ Lower(x.f = v) & = & x.f = v \\ Lower(\,\mathbf{prt}\, x\,) & = & \texttt{true} \\ Lower(\,x\, \mathbf{prt\text{–}frm}\, \overline{u}) & = & x\, \mathbf{prt\text{–}frm}\, \overline{u} \end{array}$$

...

Challenge_4: An inference system, such we can prove external calls

ExtCall

$$\frac{\begin{array}{c} M \vdash A \;\rightarrow\; (\; \mathbf{extl}\, z \;\wedge\; \mathbf{extl}\, \overline{u} \;\wedge\; \mathbf{intl}\, \overline{y} \;) \qquad \overline{v} \text{ is a permutation of } \overline{u}, \overline{y} \\ \vdash\; M \;:\; \forall \overline{x : C}.(\!|\, A_1 \,|\!)(\!|\, A_2 \,|\!) \\ M \vdash Lift(A, \{z, \overline{u}\}, \overline{y}) \;\rightarrow\; (\, \overline{x : C} \wedge A_1 \,) \end{array}}{M \;\vdash\; \{\; A \;\}\; z.m(\overline{v})\; \{\; Lower(A_2) \;\wedge\; A \;\}}$$

Challenge_4: An inference system, such we can prove external calls

ExtCall

$$\frac{\begin{array}{c} M \vdash A \;\rightarrow\; (\; \mathbf{extl}\, z \;\wedge\; \mathbf{extl}\, \overline{u} \;\wedge\; \mathbf{intl}\, \overline{y}\;) \qquad \overline{v} \text{ is a permutation of } \overline{u}, \overline{y} \\ \vdash\; M \;:\; \forall \overline{x : C}.(\!|\, A_1 \,|\!)(\!|\, A_2 \,|\!) \\ M \vdash Lift(A, \{z, \overline{u}\}, \overline{y}) \;\rightarrow\; (\,\overline{x : C} \wedge A_1\,) \end{array}}{M \;\vdash\; \{\; A \;\}\; z.m(\overline{v})\; \{\; Lower(A_2) \;\wedge\; Prsv(A, \{z, \overline{u}\}, \overline{y}, M) \;\}}$$

$$\begin{array}{lcll} Prsv(v = v', \overline{z}, \overline{y}, M) & = & v = v' & \\ Prsv(x.f = v, \overline{z}, \overline{y}, M) & = & x.f = v & \text{if } \vdash\; M \;:\; \forall \overline{x' : C}, x : D.(\!|\, x.f = v \wedge A_1 \,|\!)(\!|\, x.f = v \,|\!) \\ & & & \text{and } M \vdash Lift(A, \overline{z}, \overline{y}) \;\rightarrow\; (\,\overline{x : D} \wedge A_1\,) \\ & = & \texttt{true} & \text{otherwise} \\ Prsv(\,\mathbf{prt}\, x\,, \overline{z}, \overline{y}, M) & = & \mathbf{prt}\, x & \\ Prsv(\,x\, \mathbf{prt\text{–}frm}\, \overline{u}, \overline{z}, \overline{y}, M) & = & x\, \mathbf{prt\text{–}frm}\, \overline{u} & \text{if } \vdash\; M \;:\; \forall \overline{x' : C}, x : D.(\!|\, x.f = v \wedge A_1 \,|\!)(\!|\, x.f = v \,|\!) \\ & & & \text{and } M \vdash Lift(A, \overline{z}, \overline{y}) \;\rightarrow\; (\,\overline{x : D} \wedge A_1\,) \\ & = & \texttt{true} & \text{otherwise} \end{array}$$

Challenge_4: An inference system, such we can prove external calls

Motto:
Capability is a *necessary* condition for some effect

ExtCall

$$\frac{M \vdash A \;\rightarrow\; (\; \mathbf{extl}\, z \;\wedge\; \mathbf{extl}\, \overline{u} \;\wedge\; \mathbf{intl}\, \overline{y} \qquad \vdash \; M \;:\; \forall \overline{x : C} \qquad M \vdash Lift(A, \{z, \overline{u}\}, \overline{y}) \;\rightarrow\; (\; x : C \wedge A_1 \;)}{M \;\vdash\; \{\; A \;\}\; z.m(\overline{v})\; \{\; Lower(A_2) \;\wedge\; Prsv(A, \{z, \overline{u}\}, \overline{y}, M) \;\}}$$

$$
\begin{array}{lcll}
Prsv(v = v', \overline{z}, \overline{y}, M) & = & v = v' & \\
Prsv(x.f = v, \overline{z}, \overline{y}, M) & = & x.f = v & \text{if } \vdash \; M \;:\; \forall \overline{x' : C}, x : D.(\!|\, x.f = v \wedge A_1 \,|\!)(\!|\, x.f = v \,|\!) \\
 & & & \text{and } M \vdash Lift(A, \overline{z}, \overline{y}) \;\rightarrow\; (\; \overline{x : D} \wedge A_1 \;) \\
 & = & \texttt{true} & \text{otherwise} \\
Prsv(\, \mathbf{prt}\, x \,, \overline{z}, \overline{y}, M) & = & \mathbf{prt}\, x & \\
Prsv(\, x\, \mathbf{prt\text{–}frm}\, \overline{u}, \overline{z}, \overline{y}, M) & = & x\, \mathbf{prt\text{–}frm}\, \overline{u} & \text{if } \vdash \; M \;:\; \forall \overline{x' : C}, x : D.(\!|\, x.f = v \wedge A_1 \,|\!)(\!|\, x.f = v \,|\!) \\
 & & & \text{and } M \vdash Lift(A, \overline{z}, \overline{y}) \;\rightarrow\; (\; \overline{x : D} \wedge A_1 \;) \\
 & = & \texttt{true} & \text{otherwise}
\end{array}
$$

Challenge_4: An inference system, such we can prove external calls

$$\textsc{prot-1}\ \frac{}{M \vdash \{\ x\ \mathbf{prt\text{–}frm}\ z\ \wedge\ \mathbf{intl}\ v\ \}\ v = v'\ \{\ x\ \mathbf{prt\text{–}frm}\ z\ \}}$$

$$\textsc{prot-2}\ \frac{}{M \vdash \{\ x\ \mathbf{prt\text{–}frm}\ z\ \wedge\ \mathbf{extl}\ v\ \wedge\ z \neq v\ \}\ v = v'\ \{\ x\ \mathbf{prt\text{–}frm}\ z\ \}}$$

$$\textsc{prot-3}\ \frac{}{M \vdash \{\ x\ \mathbf{prt\text{–}frm}\ \{v\ \wedge\ \mathbf{extl}\ z\ \}\ z = v\ \{\ x\ \mathbf{prt\text{–}frm}\ z\ \}}$$

$$\textsc{prot-4}\ \frac{}{M \vdash \{\ x\ \mathbf{prt\text{–}frm}\ z\ \}\ y.f = v\ \{\ x\ \mathbf{prt\text{–}frm}\ z\ \}}$$

Challenge_4: An inference system, such we can prove external calls

PROT-1

$$\frac{}{M \vdash \{\ x\ \mathbf{prt\text{–}frm}\ z\ \wedge\ \mathbf{intl}\ v\ \}\ v = v'\ \{\ x\ \mathbf{prt\text{–}frm}\ z\ \}}$$

PROT-2

$$\frac{}{M \vdash \{\ x\ \mathbf{prt\text{–}frm}\ z\ \wedge\ \mathbf{extl}\ v\ \wedge\ z \neq v\ \}\ v = v'\ \{\ x\ \mathbf{prt\text{–}frm}\ z\ \}}$$

PROT-3

$$\frac{}{M \vdash \{\ x\ \mathbf{prt\text{–}frm}\ \{v\ \wedge\ \mathbf{extl}\ z\ \}\ z = v\ \{\ x\ \mathbf{prt\text{–}frm}\ z\ \}}$$

PROT-4

$$\frac{}{\ldots\ \}\ y.f = v\ \{\ x\ \mathbf{prt\text{–}frm}\ z\ \}}$$

Motto:
Tracking access to Capabilities

Challenge_2: An inference system, such that … — **revisit**

TWO-STATE

$$\frac{\begin{array}{c} M \vdash \mathit{Encaps}(\overline{x : C} \wedge A) \\ \text{for all public methods from } D, \text{ with } \mathtt{mBody}(m, D, M) = \overline{y : D}\{\, s \,\} \\ M \vdash \{\overline{x : C} \wedge A \wedge \mathtt{this} : \mathtt{D} \wedge \overline{y : D}\}\, s\, \{\, A \,\} \parallel A \end{array}}{M \vdash \forall \overline{x : C}.(\!|A|\!)(\!|A|\!)}$$

BODY

$$\frac{\begin{array}{c} M \vdash \{A'\}\, s\, \{\, A \,\} \\ \forall s', z, m.[\ (s = s'; z.m(\_); \_ \wedge M \vdash \{A'\}\, s'\, \{\ \mathbf{extl}\, z\ \}) \implies M \vdash \{A'\}\, s'\, \{\, A \,\}\ ] \end{array}}{M \vdash \{A'\}\, s\, \{\, A \,\} \parallel A}$$

WEAKEN

$$\frac{M \vdash S \qquad M \vdash S \sqsubseteq S'}{M \vdash S'}$$

MULTI

$$\frac{M \vdash S \qquad M \vdash S'}{M \vdash S \wedge S'}$$

Challenge_2: An inference system, such that … — **revisit**

TWO-STATE

$$\frac{\begin{array}{c} M \vdash \mathit{Encaps}(\overline{x : C} \wedge A) \\ \text{for all public methods from } D, \text{ with } \mathtt{mBody}(m, D, M) = \overline{y : D}\{\, s \,\} \\ M \vdash \{\overline{x : C} \wedge A \wedge \mathtt{this} : \mathtt{D} \wedge \overline{y : D}\}\, s\, \{\, A \,\} \parallel A \end{array}}{M \vdash \forall \overline{x : C}.(\!|A|\!)(\!|A|\!)}$$

BODY

$$\frac{\begin{array}{c} M \vdash \{A'\}\, s\, \{\, A \,\} \\ \forall s', z, m.[\ (s = s'; z.m(\_); \_ \wedge M \vdash \{A'\}\, s'\, \{\, \mathbf{extl}\, z \,\} \implies M \vdash \{A'\}\, s'\, \{\, A \,\}\ ] \end{array}}{M \vdash \{A'\}\, s\, \{\, A \,\} \parallel A}$$

WEAKEN

$$\frac{M \vdash S \qquad M \vdash S \sqsubseteq S'}{M \vdash S'}$$

MULTI

$$\frac{M \vdash S \qquad M \vdash S'}{M \vdash S \wedge S'}$$

Challenge_3: The inference system should be algorithmic

Happy!

Convinced!

Surprised!

Summary

- Distinction between external/internal objects
- Specifications talk about necessary conditions for effect:
  ∀ x: Object (| **denial of capability** ∧ A |) (| A |)
  means that **capability** is needed in order to invalidate A
- **prt** x: expresses that capability x is protected from external objects
- **prt** x only protects from *locally-relevant* objects;
  ∀ x: ..(|..|)(|…|) tallks about *globally-relevant* objects
- “future is shallow”
- Algorithmic inference system;
- Can deal with external calls
- Hand-written soundness and adherence proof

Happy!

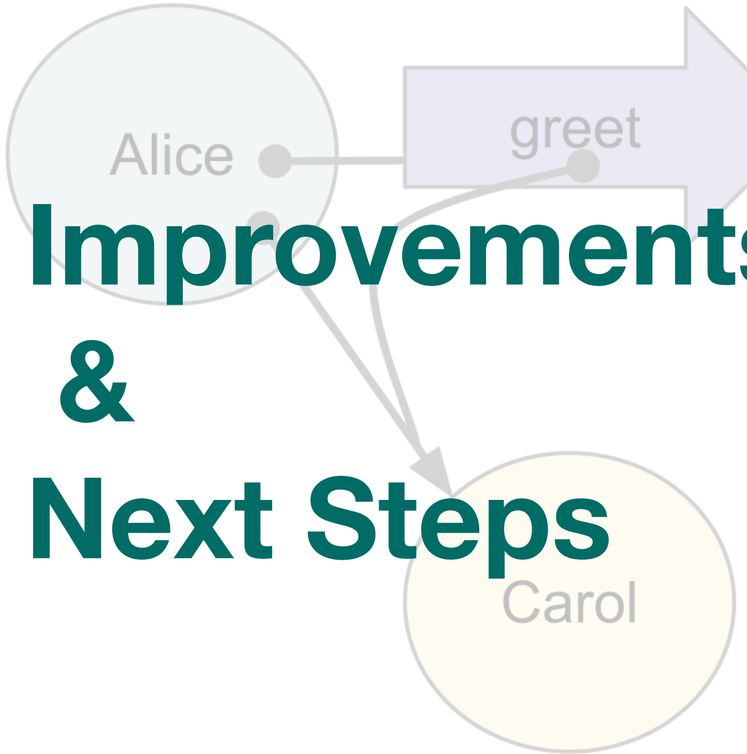
Convinced: object capabilities are about the necessary conditions for effects caused by external objects.

Surprised!

Happy!

Convinced!

Surprised: Even though we talk about *necessary* conditions, we reason with *sufficient* conditions.



Improvements & Next Steps

- Soundness machine checked
- Completeness?
- Revisit protection:
  What if more than one capability for an effect?
  protection in logical implications?
- Beyond private types: Ownership types, membranes etc
- Instance-level protection
- More than one module
- Trust and Risk
- Other programming Paradigms